The Design and Implementation of an Automated Dartboard

6.111 Final Project Report
Submitted By:

Ankush Patel and
Michael Ehrenberg

December 14th 2005

ABSTRACT

Our final project describes the creation of a dartboard that can score the popular 2 player dart games 301 and 601. Dart detection was performed by acoustic triangulation, which uses the difference in arrival times of the sound of contact made with the dartboard to determine individual dart location. Once x and y coordinates are determined they are displayed on a graphical user interface showing their position on the dartboard and indicating the status of the game. This project successfully illustrates many of the concepts that are essential to digital design and implementation discussed throughout 6.111.

TABLE OF CONTENTS

TABLE OF FIGURES

PROJECT OVERVIEW

Darts is traditionally a game that does not involve electronics. All you need is a board and a set of darts to play, and scoring is most often performed by hand. For this reason we chose to fully automate a dartboard, providing a no hassle approach to enjoying darts. On a high level, our dartboard has the ability to tell where darts have been thrown by a player and subsequently keep score according to the rules of the two player game 301 or 601.

In the dart world, automatic dartboard scoring already exists; however, it is not used in competitive play because players are looking for the simplest and most inexpensive interface. While most scoring is performed by hand, there are two different types of automation that are currently available. The first scoring device is the simpler of the two which simply keeps score but does not incorporate dart detection. A human player must enter the score for a given turn, and the electronics will do all the work of displaying the total score and status of the game. The second type of scoring device is fully automated and uses lasers or has pressure sensors; however, a board like this is quite expensive with an average price of around 400 dollars, and these boards are cumbersome and awkward looking. As a result, this project aims to advance automated dartboard scoring by minimizing the board price and reducing the bulk of the board.

Before describing the functionality of the dartboard, we must first understand the rules of 301. 301 is a simple and common dart game played between two people. After each player throws three darts, the sum of their points scored is subtracted from the number 301. The first player to reach zero wins. 601 is the same as 301 except the starting number that both players subtract from is 601. Figure 1 below displays a dartboard so that you can see how scoring is calculated. Each triangle wedge is worth the number of points listed on the outside. The outer ring is worth double that number of points, and the middle ring is worth triple that number of points. Last but not least, the inner ring otherwise known as the bullseye is worth 25 or 50 points depending on whether your dart lands in the green inner circle (25) or the red inner circle (50). There are two additional rules that are important to the game of 301. Before a player can begin subtracting points from 301, he must "double in", and this means he must throw a dart in the outer ring of the board which counts as double the score for whatever number wedge the dart lands in. Also, the player must "double out" reaching exactly zero to end the game. For example, if player one is down to 10, he must hit a double 5 to win. A double 5 is pointed to by the red arrow in the dartboard illustration below. If he hits a single 15 or any number greater then 10 instead, it is called a "bust" and he must wait until his next turn to throw again. However, if the player scores a combined 6 between his three darts that will be subtracted from his score and the player will be required to hit a double 2 on his next turn to win.

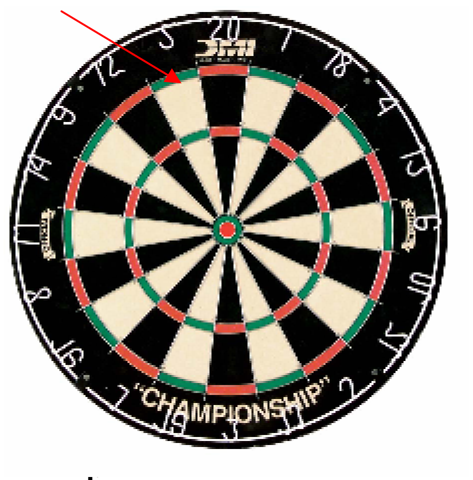

Figure 1: Dartboard

Arrow points to double 5

DART DETECTION

With this game in mind, there are two main components to our dartboard, dart detection, and a graphical game interface. First, in order to detect dart location we used acoustic triangulation. Using this method, we placed three microphones around the board. When a dart is thrown the relative amount of time it takes the sound of impact to reach the microphones will indicate the darts position using triangulation. More specifically, when the microphone closest to the dart detects impact, we count the number of clock cycles until the other microphones hear the dart impact. This difference between times allows us to convert clock cycles into a distance from the center using the speed of sound, and the frequency of the clock. In order for the computer to learn when it should start listening, the user will press a button to indicate the beginning and end of each turn.

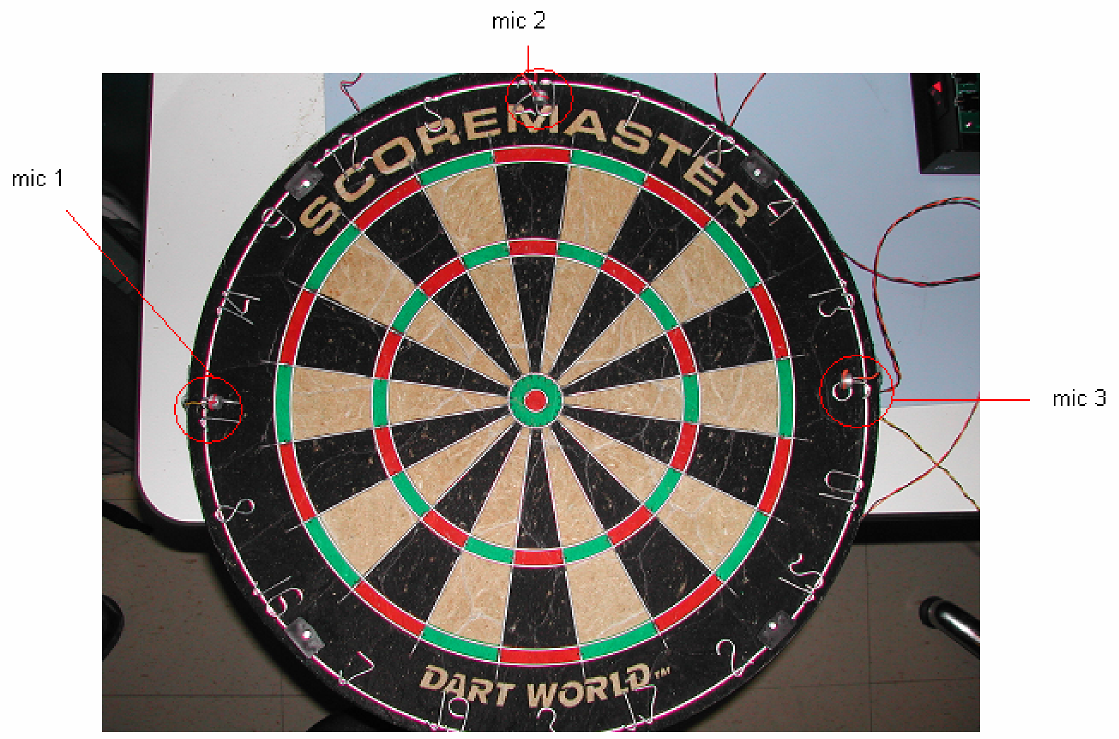

Figure 2: Dartboard with Microphones

Three microphones used for acoustic triangulation

USER INTERFACE

Once the x and y coordinates of the three darts for a given players turn are determined, our system illustrates the progress of the game on a video display. The display contains an image of a dartboard, with player 1's score to the left and player 2's score to the right. After all three darts are thrown, their calculated location is drawn on the board. Once the darts are displayed, the user has the ability to correct the darts if there are any discrepancies due to a microphone misreading or not a suitable amount of resolution. After allowing the user to correct darts, the score is updated appropriately and it becomes the next players turn. Figure 2 is a picture of the final product graphical user interface of the automatically scoring dartboard. In this image you can see the dartboard, the dart locations being displayed on the screen as well as each darts appropriate score. In addition, the individual player scores are illustrated as well as the turn indicator to tell whether it is the turn of player 1 or player 2.

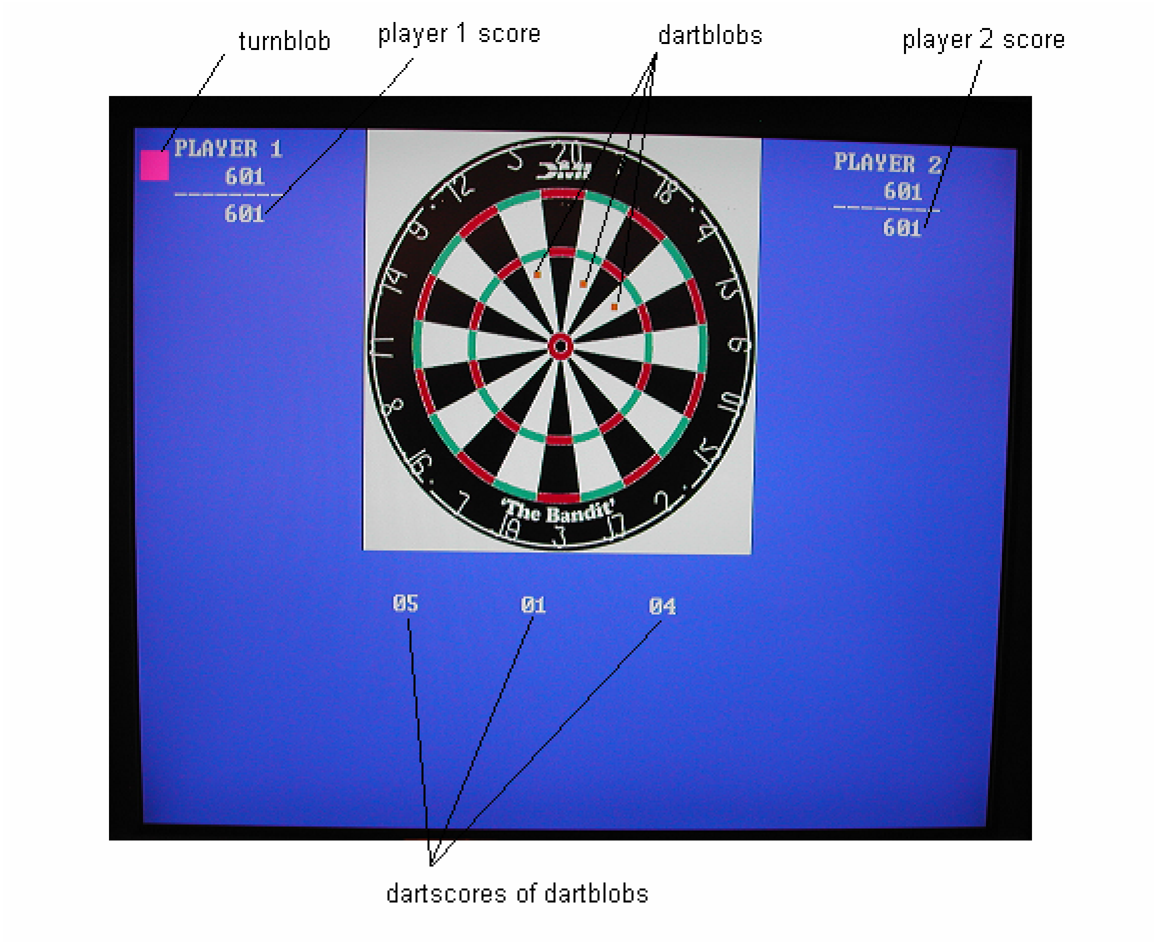

Figure 3: Graphical User Interface

The above image shows the final graphical user interface for the automated dartboard

The details of the graphical user interface and dart detection implementation are described in more detail in the next section as we describe our design in more detail.

DESCRIPTION

Triangulation Logic Introduction:

Calculation of the position of darts proved to involve a number of complex math operations. To set up the problem we modeled the dart board in a Cartesian coordinate system centered at the bull’s eye. Each of the three microphones were positioned at some offset (x0, y0, x1, y1, x2, y2) and using the differences in times of detection (d0, d1, d2) the intersection of three circles is calculated to reveal the position of the dart. More formally we have three equations

$$(d + d0)^2 = (x - x0)^2 + (y - y0)^2$$
$$(d + d1)^2 = (x - x1)^2 + (y - y1)^2$$
$$(d + d2)^2 = (x - x2)^2 + (y - y2)^2$$

and three unknowns, d (distance to closest microphone), x (x-coordinate of dart) and y (y-coordinate of dart), with the rest of the variables either being constants we control with placement of the microphones or values we measure. After substituting in values and solving we obtain for d:

$$d = \frac{-d0^3 + 80000d1 + d0^2 d1 + d0d1^2 - d1^3 + 80000d2 + d0^2 d2 + d0d2^2 - d2^3 - \sqrt{\ldots}}{2\left(-80000 + 2d0^2 + d1^2 + d2^2 - 2d0\left(d1 + d2\right)\right)}$$

Substituting this in for d and solving two equations for y results:

$$y = \frac{80000 - 2dd0 - d0^2 + 2dd1 + d1^2 - \sqrt{\left(80000 + 4d^2 + d0^2 + 2d0d1 + d1^2 + 4d\left(d0 + d1\right)\right)}}{800}$$

Finally substituting both into one equation and solving reveals:

$$x = \sqrt{\left(d + d0\right)^2 - (y - 200)^2}$$

As expected the resulting equations only rely on the differences in times of detection of the darts. Luckily many other terms drop out as well when one of d0, d1 or d2 terms is zero, signifying the associated microphone was the first to detect the noise from impact.

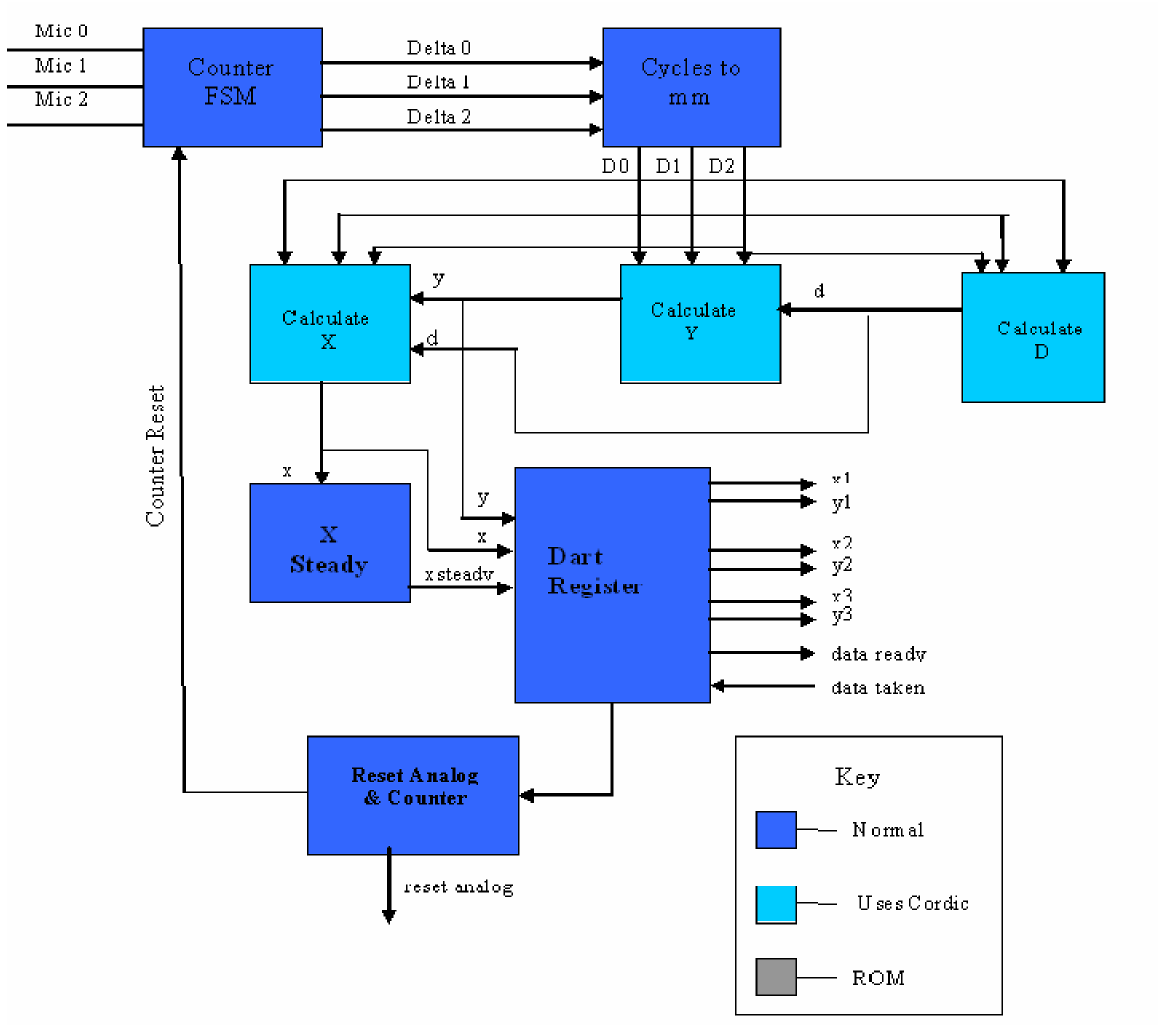

Figure 4: Dart Detection Block Diagram

Dart Detection Block Diagram Description

Counter - The first module needed in the process of dart detection is the Counter Module. The purpose of the Counter Module is to record differences in detection time between the three microphones. When a dart is thrown and impacts the board, one of the three microphones will detect the sound of impact first, triggering counts associated with the other two microphones to be started to record the time between the detection by the first microphone and the detection by the second and third microphones. Once the second microphone detects the impact its count is stopped, still incrementing the last count until the last microphone detects the sound of impact. Counts are incremented at each 2.7mhz enable provided by the divider module. This module is implemented as a Finite State Machine (fsm) with eight stages, one reset state, one ready state and six states corresponding to each combination of counts, for example mic1 count state and mic1 & mic2 count state. A ready signal is also established to alert other modules that new time differences and eventually distances are available.

Convert cycles to millimeters - In order to convert differences in cycles into distances, we utilize the speed of sound in air and the clock cycle rate:

$$340000\frac{mm}{s}\times\frac{1s}{2.7\times10^{6}\,cycles}=.125\frac{mm}{cycle}$$

The Cycles to Millimeters Module utilizes this relation to calculate the distances associated with the microphones. Fortunately, every eight clock cycles corresponds to 1mm traveled. Thus the calculation simply becomes a shift by 3 bits and no complicated multiplication/division circuitry is needed. This explains why a seemingly arbitrary 2.7 MHz sample rate was used. From here the three distances are inputted into the three major math modules.

Calculate Modules - The first math module used in the calculation of the position of the dart is the Calculate D module. This module implements the equation written above through a cascaded Cordic square rooters and multipliers. Once this value is ready it is passed to Calculate Y Module which in turn implements the second equation above. This module also uses many of the provided Cordic square rooters and multipliers to calculate the y value of the dart. Again this value is fed forward to the Calculate X module where the value of y and d are used to finally find the value of x.

X Steady - In order to assure that the value provided by the Calculate X module is valid another module was implemented. The purpose of the X Steady Module is to wait through enough clock cycles to ensure that the latest ready signal provided by the Calculate X module is actually for the latest data provided by the counter module and not for some intermediate values calculated along the way.

Dart Register & Reset Analog - Once it is known that the values calculated by the X and Y modules are correct a module called the Dart Register saves the values of x and y, so they can be provided to the display logic in three-dart pairs. After each dart is stored a reset signal is sent to the counter module so it can begin to detect new darts. Also, the latches in the analog circuitry amplifying the signals from the microphones must also be reset. Unfortunately, the analog circuitry cannot be reset with a signal for only one clock cycle. For this reason the Reset Analog & Counter Module provides a reset signal to the counter module and the latches for one third of a second. In practice a third of a second proves enough to consistently reset the latches and the chances of all three darts arriving in the same second is nearly impossible if the darts are thrown sequentially. Once three sets of x and y coordinates are provided to the dart register a data ready signal is asserted so that the display logic can take the three saved pairs and display them on the screen. Once the signal data taken is asserted the dart register knows to reset all of the logic in the dart detection part of the system. Thus the counter will begin to detect darts again and the dart register will clear the previous three saved darts coordinates.

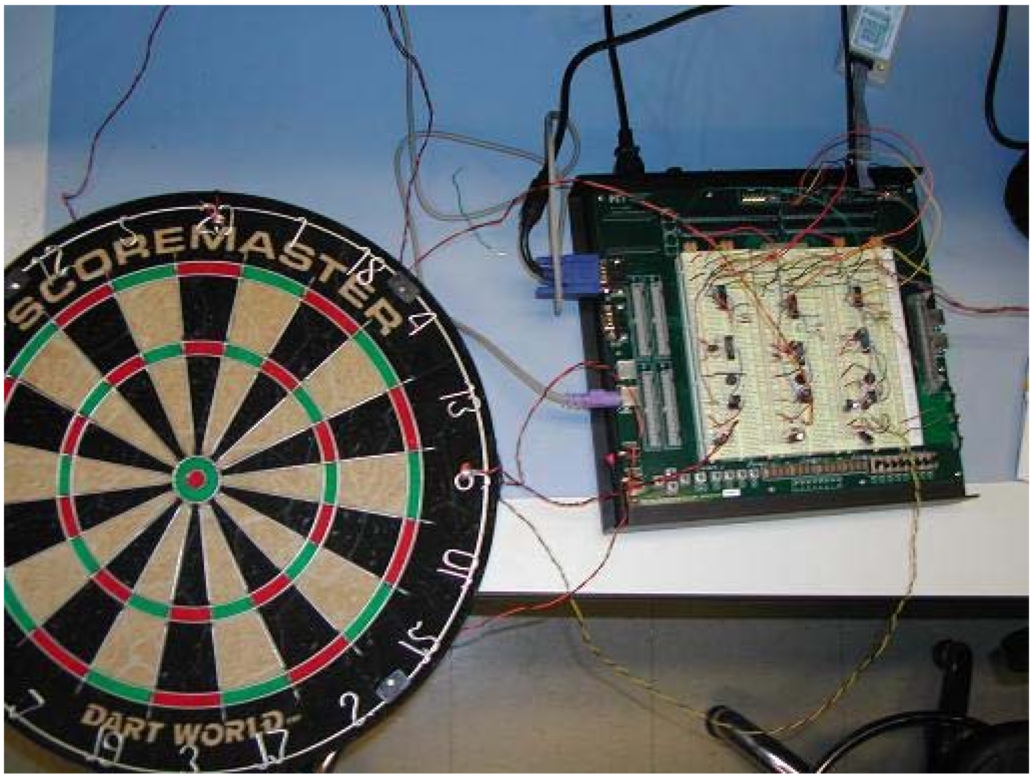

Figure 5: Dart Detection Hardware

The breadboards in this image are the circuitry for microphone amplification. You can also see the connections between the microphones on the dartboard to the labkit circuitry

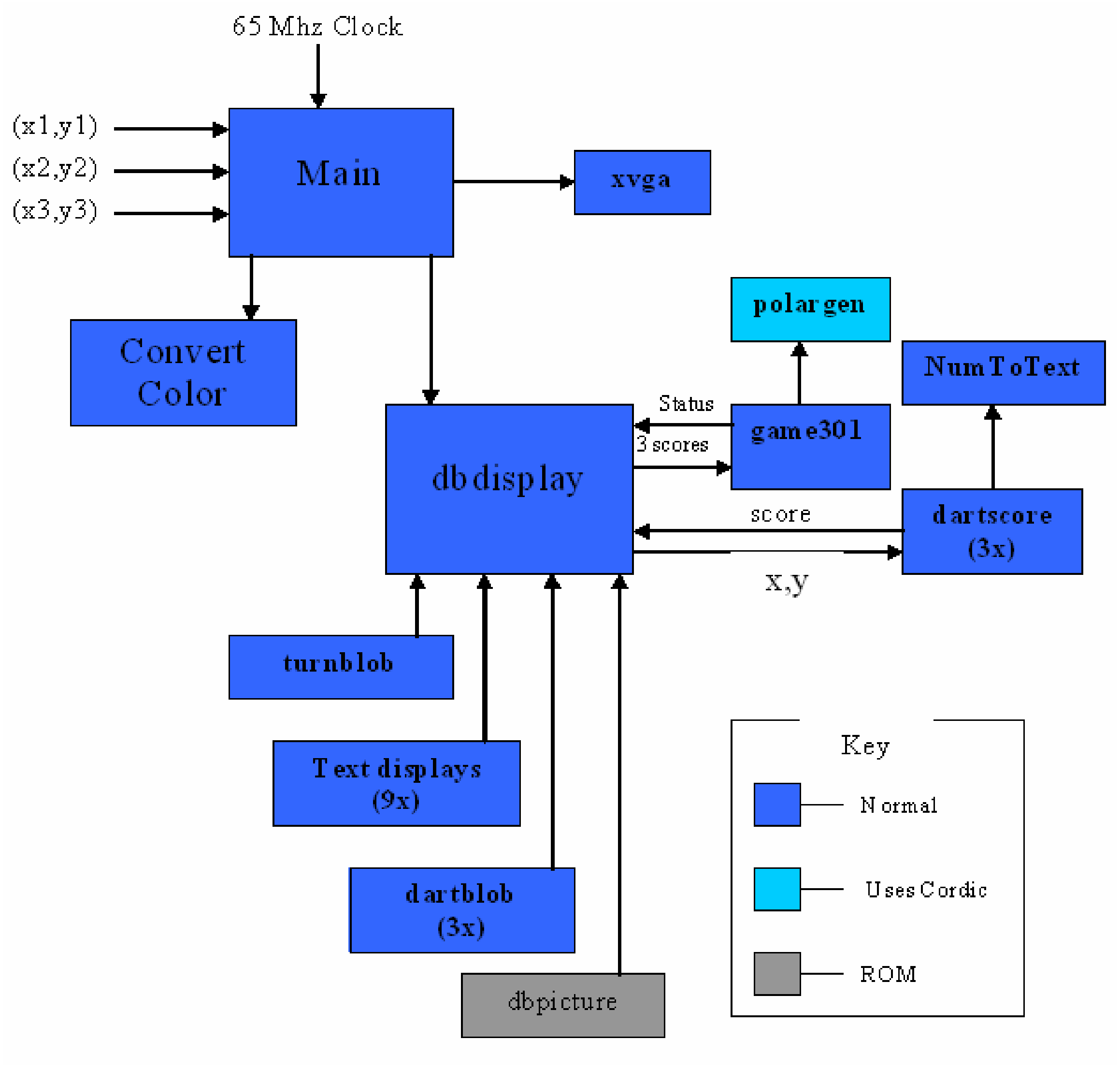

Figure 6: User Interface Block Diagram

GUI Block Diagram Description

The graphical user interface is made of several components. Each time that a player throws three darts, the x and y coordinates of each are fed into the interface modules. The user interfaced modules run on a 65 Mhz clock, which differs from the dart detection clocks. Since only a few values are transferred, and they are all transferred simultaneously we were able to avoid integration bugs by simply latching the data in the dart detection module before sending it to the interface modules.

Once dart data has been retrieved it is important to look in more detail at the individual modules that display the status of the dart game on the monitor.

dbdisplay – this is the most important module of the interface. Dbdisplay in addition to accepting as input the dart values displays all of the images and text shown on the screen. The first and most important of those images is the dartboard. The dartboard is a 453x453 image read from a ROM. The size 453x453 is significant because it is the size of an actual dartboard, and by displaying an image with similar dimensions a conversion was avoided from the dart detection coordinates to the screen coordinates. Also, dart blobs which illustrate dart location, a turn blob which is an indicator that illustrates whether it is player 1 or player 2 who is next, and screen text are all generated in this module. Finally dart correction is performed in this module. Dart correction is performed by moving dart blobs based on the arrow keys on the 6.111 labkit, and then re-calculating the score of a dart.

dartblob – Based on hcount and vcount, dartblob indicates whether pixels indicating a dart should be displayed in the dbdisplay module.

turnblob – Based on hcount and vcount, turnblob indicates whether pixels indicating whether player1 should be highlighted with a red square next to the player one score or player two should be highlighted with a red square next to the player two score.

dartscore – This module accepts the x and y value of an individual dart and returns the appropriate ascii score from 1 – 60 or indicates "NO SCORE" if the dart is off of the board. Dartscore performs this operation by using a *cordic* module to convert the x and y coordinates to polar coordinates, r and theta. The value r is generated by taking the squareroot of $x^2 + y^2$ and theta is determined by taking the inverse tangent of y divided by x. Once this conversion has been made, theta is used to determine which pie shaped region the dart is in between 1 and 20. Next r is used to determine if the dart is within the single bullseye, double bullseye, single score, double score, or triple score. This number is returned as an ascii string to make it easier to display.

game301 – This module acts as either 301 or 601 based on a switch on the labkit. The two games can be interchanged only before the start when both players still have a score of 301 or 601 and can be changed unless the game is restarted. This module accepts the scores of the three darts as ascii strings when all dart modifications have been made and the user presses button zero on the labkit. The game301 module then converts the ascii strings to binary, adds up the scores, and determines which score to subtract from. This module also includes logic to include the double in and double out rules. Once this has been determined, the final binary score is then converted back to ascii by using a simple lookup table that has all of the numbers from 1 to 601 listed and returns to the dbdisplay.

convertcolor – this module is a lookup table necessary to display the proper colors for the dartboard image on the screen. The dartboard image utilized in this project is an 8 bit bitmap. Therefore each pixel contains 8 bits of color information. 8 bits means that there are 256 different possible colors. Unfortunately these 8 bits do not correspond directly to a red green and blue color value but instead refer to an index in a color palette. As a result, when a pixel of the dartboard image is retrieved an 8 bit index number is given and the main module looks at the convertcolor module to find the exact ratio of red green and blue at the given index value in the color palette.

textdisplays – There are 9 different text displays on the user interface. Each has its own text module which has a position on the screen. These text display modules are combined with a bitwise OR in the dbdisplay module to correctly illustrate them.

TESTING AND DEBUGGING

Dart Detection Testing and Debugging

The first stage of debugging for the detection part of the project involved the microphones and the corresponding analog circuitry. After the circuitry was built up using cascaded op amps, BJT's and nand-gate latches we had to make sure that the output was reasonably related to the real world sound input. In order to do so we connected an oscilloscope to various points along the circuit's path. To begin the output after the last op amp was monitored and by providing sudden large impulses of sound, snapping of the fingers, while also providing more monotone signals, like whistling. Next we needed the output voltage to be TTL compliant so passed the output through a BJT and again monitored the output on the oscilloscope. This time we expected and found that the output would be a series of square waves with peaks at 5V and lows at 0V. Now the output is ready to pass through cascaded nand-gates to form a latch. The output from the latch, which we later inputted into the labkit, was pretty remarkable as well. Given an input, usually snapping of the fingers, to a microphone the output of the latch viewed on the oscilloscope was always a perfect step function. To test that there wasn't a significant delay in this logic two channels of the oscilloscope are used to display the output of the op amp and the output of the latch. These two signals were perfectly synchronized proving that the circuitry not only functioned correctly but also very efficiently. Except for the occasional burnt out chip this circuitry worked well and was consistent throughout the latter part of the project.

After the inputs from the microphone circuitry were inputted into the labkit, the first module that needed to be tested was the counter module. The most convenient method to do this utilized both the oscilloscope and the logic analyzer. The oscilloscope was used to monitor the actual output and differences in detection times of the three latches. A one mhz enable signal was provided to the counter module so the counts outputted by the module would correspond exactly to the output on the oscilloscope (both would be in microseconds). The output of the counter module was wired to the pins for the logic analyzer so the values from the module could be compared to the actual differences. At first a weird "bug" emerged as the counter module counted exactly twice as many microseconds as the oscilloscope measured. It turns out that the logic analyzer was connected improperly to the labkit, providing a shift by one bit to the analyzer. In retrospect, it may have been better to use a different output rather than the logic analyzer to test this module, such as the hex-display, since using the logic analyzer in this manner seemed quite wasteful. After a reasonable amount of testing using this method it became clear that the counter module worked correctly.

The cycles to millimeters module was implemented by shifting by 3 bits. For this reason I found it adequate to use a simple test bench waveform to test the module and only come back to this module if later results seemed incorrect. The test bench proved that the module would do the simple shift and provide an approximate division by 8 as required for the rest of the logic in the system.

The next three modules proved to be the most difficult to implement and test. The calculations of d, y and x were done in separate modules and thus tested independently. First specific values for the inputs to the calculate d module (d0, d1, d2) were chosen where the resulting d is known. Then the output of the module was connected to the hex-display on the labkit to compare the exact values expected to the actual values calculated by the logic. In almost every case the values calculated were only one millimeter off from the expected value. This one millimeter discrepancy was due to the truncation of floating point numbers in the multiplication, square root and division modules. After d was verified as correct, the same method was repeated to test the calculate y and then the calculate x module. In the end switches on the labkit were used to load in different values for d0, d1, d2 and the output was again displayed on the hex-display. Once satisfied with the results for constant values, the three modules were connected to the rest of the system to test integration with the counter module. To make sure all the modules were working correctly together, the latched output of the microphones were connected to the oscilloscope again. Using the oscilloscope values for d0, d1, d2 can be observed and the appropriate x and y coordinates can be calculated. The outputs x and y are once again connected to the hex-display on the labkit and the calculated values are compared to the displayed values. If there are any discrepancies values for d and d0, d1 and d2 can be displayed to find the root of the problem.

The last module to be implemented and tested was the dart register. This module provides the interface between the dart detection and dart display portions of the project. To ensure proper functionality of this module three darts were thrown at the dart board and their calculated positions were recorded. Switches on the labkit choose which of the three darts stored in the module were displayed on the hex-display. If the recorded values matched the displayed values the module is functioning correctly. Unfortunately, in order to test this module the automatic reset of the latches and counter had to be replaced with reset on a button press. In the end the automatic reset was returned and the register was connected to the dart display logic.

User Interface Testing and Debugging

In order to test the graphical user interface two methods were used. For the text displays, the dartboard image display, the dartblobs, the turnblob and dart scoring I used the output on the screen to debug. In contrast for the game301, numtotext, and polargen, I used the test bench wave forms to guarantee correct values. During integration, the hex display was used to ensure that the values that the user interface displayed were the same as the values that the microphone was providing.

First, the dartboard image display was the first component that I added to the user interface. I formatted it in an 8 bit format and then created a *.coe file which was loaded into a rom. In order to test this I made sure that the image on the screen was the same as the image in the image file that I used to create the *.coe file. Once I initially was able to get the image to display there were several problems with the coloring. In order to correct this I created a color palette which is found in the convertcolor module and this corrected the bad coloration.

Next I added the text displays to the interface. These were easy to test because I needed to ensure that they displayed what I intended them to display on the screen.

Once I could display text, I added the dartblobs to the screen. I set up sample coordinates for three different darts and fed that into the dbdisplay module. One dart I assigned x, y coordinates of (0,0) which is a bullseye one I assigned at the outer edge of the board (170, 170) and the other I put at a random. By simply looking at the output screen it was easy to tell when I had the correct locations.

One of the more challenging modules to test was the 301module.   The 301 module includes all of the game logic for 301 and 601. I used this testbench to insure

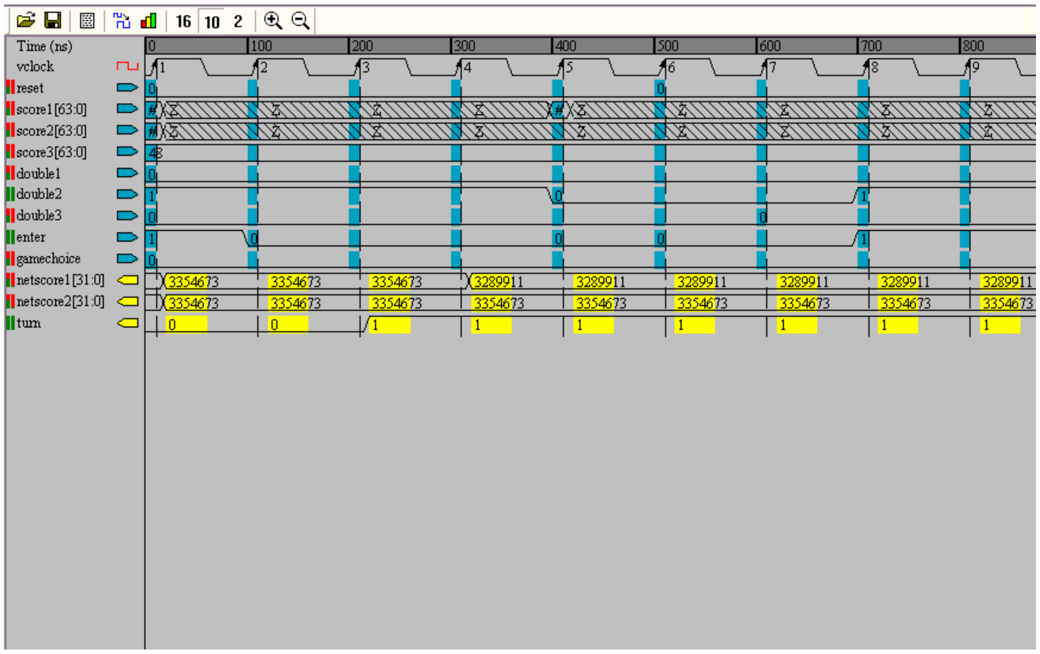

Figure 7: game301 module test bench

Test bench waveform used to test and debug the 301 game module

double in and double out worked properly. I did this by making sure scores only changed when a double was high in the beginning. I also used this test bench to make sure the scores were being updated appropriately based on the correct player turns. Another interesting problem that the test bench helped me to solve was converting a score from ascii to binary. Out of my entire test bench use for different modules, my 301module testbench was the most important and the most helpful.

Another place where I spent a lot of time testing and debugging was the dart score module. In this instance I used both the test bench waveform as well as the output of a given darts score to the screen. I used the testbench to make sure that the values I received from the x, y to polar coordinates conversion were correct, and then by looking at the screen it was simple to determine if a dartblob in a certain scoring area gave the correct output.

Integration Testing

To test how well the system integrated we physically threw darts at the board and compared expected values calculated using the oscilloscope and a calculator program to labkit calculated values. Furthermore, we compared the outputted x and y values from the labkit which we outputted to the hex display to placement on the screen and ensured that they were accurately placed

CONCLUSION

Building an automated dartboard was an exciting idea when we first thought of its possibilities, and after the completion of our project it still is exciting because our automated dartboard accomplished nearly all of our original goals. In addition, we were able to compensate for several of the boards weaknesses by implementing several nice to have features that were not considered must haves for the success of this project, but which significantly contribute to the game experience.

Originally, we set out to create a dartboard that was not as bulky and expensive as current boards that automatically score a competitive darts match. Through our design, we achieved this by using an unmodified dartboard and simply adding three small microphones around the outside, and extending wires to the labkit. This non-obtrusive hardware element does not interfere with the game and is hardly noticeable. We are confident that our approach to automating a dartboard would cost significantly less then $400 if commercially implemented due to the cheap cost of microphones as opposed to lasers or pressure sensors.

We also exceeded our goals by implementing nice to have features such as dart correction and the additional game of 601. Dart correction is extremely important, especially, when the microphones did not give the values that were expected. In these circumstances dart correction allows the user to move the darts on the user interface into their correct scoring region regardless of what the hardware determined as the value. In addition by programming the game of 601 in addition to 301 our project offers a wider variety of game play which allows the dartboard to appeal to a wider audience.

Unfortunately, with all of the success we had with this dartboard we were not able to achieve a dart resolution of less then one inch. This lack of resolution is due to several factors. First the microphones that we used were not always reliable and often gave incorrect results. In hindsight we should have spent more money to get a better set of microphones that did not require us to build the amplification circuitry ourselves. Also, in our design we chose to specific locations for our microphones along the x and y axis and truncated a lot of the floating point values during mathematical operations, both of which we estimate reduced our resolution by several millimeters. One possible way that we could have avoided this reduced resolution would rely on a successful implementation of a lookup table that gives x, y coordinates for each set of varied clock differences in the triangulation calculations. We chose not to use this approach due to limited memory resources, but assuming this was not a problem it would be a nice addition. Also, for our calculations we used the speed of sound in air to determine dart location, by calibrating the microphones to use the speed of sound in cork our resolution could possibly be further enhanced.

As a whole, this project was still very much a success. We compensated for weaknesses in our implementation by adding additional nice to have features and as a result our final project accomplished our primary objective to produce an automated dartboard that allows two players to play 301. The tradeoffs that were made in our design and implementation were made with the best interests of the final product in mind. Weighing the importance of different tradeoffs was the most educational part of this project for our team and as a result we will both take these improved skills with us to future design initiatives.

APPENDIX

```
///////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module (video version)
//
///////////////////////////////////////////////////////////////////////////////

module debounce (reset, clock_65mhz, noisy, clean);
   input reset, clock_65mhz, noisy;
   output clean;

   reg [19:0] count;
   reg new, clean;

   always @(posedge clock_65mhz)
     if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != new) begin new <= noisy; count <= 0; end
     else if (count == 650000) clean <= new;
     else count <= count+1;

endmodule

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
```

```
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

module finalproject   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
                ac97_bit_clock,

                vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
                vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
                vga_out_vsync,

                tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
                tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
                tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

                tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
                tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
                tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
                tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

                ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
                ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

                ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
                ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

                clock_feedback_out, clock_feedback_in,

                flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                flash_reset_b, flash_sts, flash_byte_b,

                rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

                mouse_clock, mouse_data, keyboard_clock, keyboard_data,

                clock_27mhz, clock1, clock2,

                disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_in,

                button0, button1, button2, button3, button_enter, button_right,
                button_left, button_down, button_up,

                switch,

                led,

                user1, user2, user3, user4,

                daughtercard,

                systemace_data, systemace_address, systemace_ce_b,
                systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

                analyzer1_data, analyzer1_clock,
                                analyzer2_data, analyzer2_clock,
                                analyzer3_data, analyzer3_clock,
                                analyzer4_data, analyzer4_clock);

                output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
                input  ac97_bit_clock, ac97_sdata_in;

                output [7:0] vga_out_red, vga_out_green, vga_out_blue;
                output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
                           vga_out_hsync, vga_out_vsync;

                output [9:0] tv_out_ycrcb;
                output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
                           tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
                           tv_out_subcar_reset;

                input  [19:0] tv_in_ycrcb;
                input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
                           tv_in_hff, tv_in_aff;
                output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
                           tv_in_reset_b, tv_in_clock;
                inout  tv_in_i2c_data;

                inout  [35:0] ram0_data;
                output [18:0] ram0_address;
                output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
                output [3:0] ram0_bwe_b;

                inout  [35:0] ram1_data;
                output [18:0] ram1_address;
                output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
                output [3:0] ram1_bwe_b;

                input  clock_feedback_in;
                output clock_feedback_out;

                inout  [15:0] flash_data;
                output [23:0] flash_address;
                output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
                input  flash_sts;

                output rs232_txd, rs232_rts;
                input  rs232_rxd, rs232_cts;

                input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
                input  clock_27mhz, clock1, clock2;

                output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
                input  disp_data_in;
                output  disp_data_out;

                input  button0, button1, button2, button3, button_enter, button_right,
                           button_left, button_down, button_up;
                input  [7:0] switch;
                output [7:0] led;

                inout [31:0] user1, user2, user3, user4;

                inout [43:0] daughtercard;

                inout  [15:0] systemace_data;
                output [6:0]  systemace_address;
                output systemace_ce_b, systemace_we_b, systemace_oe_b;
                input  systemace_irq, systemace_mpbrdy;

                output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
                                     analyzer4_data;
                output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

                ////////////////////////////////////////////////////////////////////////////
                //
                // I/O Assignments
                //
                ////////////////////////////////////////////////////////////////////////////

                // Audio Input and Output
                assign beep= 1'b0;
                assign audio_reset_b = 1'b0;
                assign ac97_synch = 1'b0;
                assign ac97_sdata_out = 1'b0;
                // ac97_sdata_in is an input

                // Video Output
                assign tv_out_ycrcb = 10'h0;
                assign tv_out_reset_b = 1'b0;
                assign tv_out_clock = 1'b0;
                assign tv_out_i2c_clock = 1'b0;
                assign tv_out_i2c_data = 1'b0;
                assign tv_out_pal_ntsc = 1'b0;
                assign tv_out_hsync_b = 1'b1;
                assign tv_out_vsync_b = 1'b1;
                assign tv_out_blank_b = 1'b1;
                assign tv_out_subcar_reset = 1'b0;

                // Video Input
                assign tv_in_i2c_clock = 1'b0;
                assign tv_in_fifo_read = 1'b0;
                assign tv_in_fifo_clock = 1'b0;
                assign tv_in_iso = 1'b0;
                assign tv_in_reset_b = 1'b0;
                assign tv_in_clock = 1'b0;
                assign tv_in_i2c_data = 1'bZ;
                // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
                // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

                // SRAMs
                assign ram0_data = 36'hZ;
                assign ram0_address = 19'h0;
                assign ram0_adv_ld = 1'b0;
                assign ram0_clk = 1'b0;
                assign ram0_cen_b = 1'b1;
                assign ram0_ce_b = 1'b1;
                assign ram0_oe_b = 1'b1;
                assign ram0_we_b = 1'b1;
                assign ram0_bwe_b = 4'hF;
                assign ram1_data = 36'hZ;
                assign ram1_address = 19'h0;
                assign ram1_adv_ld = 1'b0;
                assign ram1_clk = 1'b0;
                assign ram1_cen_b = 1'b1;
                assign ram1_ce_b = 1'b1;
                assign ram1_oe_b = 1'b1;
                assign ram1_we_b = 1'b1;
                assign ram1_bwe_b = 4'hF;
                assign clock_feedback_out = 1'b0;
                // clock_feedback_in is an input

                // Flash ROM
                assign flash_data = 16'hZ;
                assign flash_address = 24'h0;
                assign flash_ce_b = 1'b1;
                assign flash_oe_b = 1'b1;
                assign flash_we_b = 1'b1;
                assign flash_reset_b = 1'b0;
                assign flash_byte_b = 1'b1;
                // flash_sts is an input
                // RS-232 Interface
                assign rs232_txd = 1'b1;
                assign rs232_rts = 1'b1;
                // rs232_rxd and rs232_cts are inputs

                // PS/2 Ports
                // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

                // LED Displays
                //assign disp_blank = 1'b1;
                //assign disp_clock = 1'b0;
                //assign disp_rs = 1'b0;
                //assign disp_ce_b = 1'b1;
                //assign disp_reset_b = 1'b0;
                //assign disp_data_out = 1'b0;
                // disp_data_in is an input

                // Buttons, Switches, and Individual LEDs
                //lab3 assign led = 8'hFF;
                // button0, button1, button2, button3, button_enter, button_right,
                // button_left, button_down, button_up, and switches are inputs

                // User I/Os
                //assign user1 = 32'hZ;
                //assign user2 = 32'hZ;
                assign user3 = 32'hZ;
                assign user4 = 32'hZ;

                // Daughtercard Connectors
                assign daughtercard = 44'hZ;

                // SystemACE Microprocessor Port
                assign systemace_data = 16'hZ;
                assign systemace_address = 7'h0;
                assign systemace_ce_b = 1'b1;
                assign systemace_we_b = 1'b1;
                assign systemace_oe_b = 1'b1;
                // systemace_irq and systemace_mpbrdy are inputs

                // Logic Analyzer
                assign analyzer1_data = 16'h0;
                assign analyzer1_clock = 1'b1;
                //assign analyzer2_data = 16'h0;
                assign analyzer2_clock = 1'b1;
                assign analyzer3_data = 16'h0;
                assign analyzer3_clock = 1'b1;
                assign analyzer4_data = 16'h0;
                assign analyzer4_clock = 1'b1;

                ////////////////////////////////////////////////////////////////////////////
                //
                // lab4 : a simple pong game
                //
                ////////////////////////////////////////////////////////////////////////////

                // use FPGA's digital clock manager to produce a
                // 65MHz clock (actually 64.8MHz)
                wire clock_65mhz_unbuf,clock_65mhz;
                DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
                // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
                // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
                // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
                // synthesis attribute CLKIN_PERIOD of vclk1 is 37
                BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

                // power-on reset generation
                wire power_on_reset;    // remain high for first 16 clocks
                SRL16 reset_sr (.D(1'b0), .CLK(clock_65mhz), .Q(power_on_reset),
                                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
                defparam reset_sr.INIT = 16'hFFFF;

                /////////////////////////////////////////////////////
                // Mike's Initialization code
                /////////////////////////////////////////////////////

                // ENTER button is user reset
                wire reset,user_reset;
                debounce db1(power_on_reset, clock_65mhz, ~button_enter, user_reset);
                assign reset = user_reset | power_on_reset;

                // UP, DOWN, LEFT, and RIGHT  buttons for dart correction
                wire up,down,left,right, b0;
                debounce db2(reset, clock_65mhz, ~button_up, up);
                debounce db3(reset, clock_65mhz, ~button_down, down);
                debounce db4(reset, clock_65mhz, ~button_left, left);
                debounce db5(reset, clock_65mhz, ~button_right, right);

                //BUTTON0 used to add up score of 3 darts and subtract from a players score
                debounce db6(reset, clock_65mhz, ~button0, b0);

                 //BUTTON3 used by koosh to turn on hex display
                 debounce db7(reset, clock_65mhz, ~button3, button_3);
```

```
// generate basic XVGA video signals
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   xvga xvga1(clock_65mhz,hcount,vcount,hsync,vsync,blank);

   // feed XVGA signals to user's pong game
   wire [7:0] pixel;
   wire phsync,pvsync,pblank;


   reg [7:0] rgb;
   wire [7:0] red, green, blue;  //8 bits of color info for r, g, and b limited however by
colorpalette in convert color module

   convertcolor cnvrt(clock_65mhz, rgb, red, green, blue);  //color palette lookup table
   reg b,hs,vs;
   always @(posedge clock_65mhz) begin
           hs <= phsync;
           vs <= pvsync;
           b <= pblank;
           rgb <= pixel;
   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_65mhz.
   assign vga_out_red = red;
   assign vga_out_green = green;
   assign vga_out_blue = blue;
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_blank_b = ~b;
   assign vga_out_pixel_clock = ~clock_65mhz;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;


   //////////////////////////////
   // Koosh's Initialization code
   ///////////////////////////////


   wire latch_reset;  //output used to reset analog circuitry
   assign user1 = {latch_reset, 31'b0};
   assign user2 = 32'hZ;
   //inputs from outside world wired to outputs from latches
   wire mic1, mic2, mic3;
   assign mic1 = user2[31];
   assign mic2 = user2[30];
   assign mic3 = user2[29];

   wire mhz_enable;
   //divider provides 2.7mhz signal for counter module
   divider my_divider(clock_27mhz, mhz_enable);
   //differences in cycles for microphones
   wire [11:0] delta1, delta2, delta3;
   //ready and reset signals
   wire counter_ready, counter_reset, reset_lc, x_steady, x_ready, data_ready, data_taken;
   //differences in distance for microphones
   wire [8:0] d0, d1, d2;
   //input and output for dart register
   wire signed [8:0] y, x, x1, y1, x2, y2, x3, y3, steady_x, steady_y;
   //value of d
   wire [9:0] d;
   //number of darts in dart_register
   wire [1:0] dart_count;

   //this module holds the latches and counter in reset for 1/3 secs
   analog_latch_reset auto_reset(clock_27mhz, reset_lc, latch_reset, counter_reset);

   //counter counts cycles between mic detection
   counter count_fsm(clock_27mhz, mhz_enable, counter_reset, mic1, mic2, mic3, delta1,
delta2, delta3, counter_ready);

   //converts cycles to distance by shifting bits
   cycles_to_mm convert_cycles(delta1, delta2, delta3, d0, d1, d2);

   //calculates d, distance to closest microphone, given three differences in distances
   calc_d calculate_distance(d0, d1, d2, d, clock_27mhz);

   //calulated y position of dart
   calc_y calculate_y_coord(d0, d1, d2, d, y, clock_27mhz);

   //calculates x position of dart
   calc_x calculate_x_coord(d0, d1, d2, d, y, x, clock_27mhz, x_ready);

   //provides ready signal for x value, ensuring x is not changing
   steady_x_coord is_x_steady(clock_27mhz, counter_ready, x_ready, x, y, x_steady,
steady_x, steady_y);

   //holds three darts in registers and asserts data_ready when three darts are stored
   dart_register dart_reg(clock_27mhz, x_steady, x, y, x1, y1, x2, y2, x3, y3, reset_lc,
data_ready, data_taken, dart_count);

   //leds tied to mics to show detection of sound
   assign led[0] = ~mic1;
   assign led[1] = ~mic2;
   assign led[2] = ~mic3;
   assign led[3] = ~counter_ready;
   assign led[4] = ~data_ready;
   //lets us know how many darts are currently stored in register
   //helps recognize noise or missed darts
   assign led[7:6] = ~dart_count;

   //debug output from display logic
   wire signed [9:0] outx1, outx2, outx3, outy1, outy2, outy3;
   //assign 4 digits of the hex display at a time to different values
   wire signed [63:0] hex_out;
   assign hex_out[63:48] = steady_x;
   assign hex_out[47:32] = steady_y;
   assign hex_out[31:16] = outx3;
   assign hex_out[15:0] = outy3;


   display_16hex hexdisp1(button_3, clock_27mhz, hex_out,
                     disp_blank, disp_clock, disp_rs, disp_ce_b,
                     disp_reset_b, disp_data_out);




   ////////////////////////////////////////////////////
   // Mike's code
  ////////////////////////////////////////////////////


//PRIMARY DISPLAY MODULE
   dbdisplay boarddisplay(clock_65mhz,reset,b0,up,down,left, right,switch[7:5],
                     hcount,vcount,hsync,vsync,blank,
                     phsync,pvsync,pblank,pixel, switch[0], x1, y1, x2, y2, x3, y3,
                     data_ready, data_taken, outx1, outx2, outx3, outy1, outy2, outy3);


endmodule

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

```
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)
//
////////////////////////////////////////////////////////////////////////////////

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output         vsync;
   output         hsync;
   output         blank;

   reg      hsync,vsync,hblank,vblank,blank;
   reg [10:0]      hcount;    // pixel number on current line
   reg [9:0] vcount;       // line number

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   wire      hsyncon,hsyncoff,hreset,hblankon;
   assign    hblankon = (hcount == 1023);
   assign    hsyncon = (hcount == 1047);
   assign    hsyncoff = (hcount == 1183);
   assign    hreset = (hcount == 1343);

   // vertical: 806 lines total
   // display 768 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
   assign    vblankon = hreset & (vcount == 767);
   assign    vsyncon = hreset & (vcount == 776);
   assign    vsyncoff = hreset & (vcount == 782);
   assign    vreset = hreset & (vcount == 805);

   // sync and blanking
   wire      next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
```

```
  end
endmodule

///////////////////////////////////////////////////////////////////////////////
//
// dbdisplay: Main display module, puts text, images and darts on screen
// also performs dart correction
//
// Implementation By Michael Ehrenberg
//
//
///////////////////////////////////////////////////////////////////////////////

module dbdisplay (vclock,reset,enter, up,down, left, right,dartcorrection,
                  hcount,vcount,hsync,vsync,blank,
                  phsync,pvsync,pblank,pixel,gamechoice,
                  kooshx1, kooshy1, kooshx2, kooshy2, kooshx3,
                  kooshy3, data_ready, data_taken, x1, x2, x3, y1, y2, y3);

  input vclock;         // 65MHz clock
  input reset;          // 1 to reset module
  input enter;          // maps to button0 - indicates when to combine all three darts and
subtract from score
  input [10:0] hcount; // horizontal index of current pixel (0..1023)
  input [9:0] vcount;           // vertical index of current pixel (0..767)
  input hsync;          // XVGA horizontal sync signal (active low)
  input vsync;          // XVGA vertical sync signal (active low)
  input blank;          // XVGA blanking (1 means output black pixel)
  input up;             // 1 when correct dart upwards
  input down;           // 1 when correct dart downwards
  input left;           // 1 when correct dart left
  input right;          // 1 when correct dart right
  input [2:0] dartcorrection;   // indicates which dart to correct:  100 - dart 1, 010 - dart 2,
001 - dart 3
  input gamechoice;             //301 = 0, 601 = 1
  input signed [8:0] kooshx1, kooshx2, kooshx3, kooshy1, kooshy2, kooshy3;   //dart
inputs from hardware
  input data_ready;             //ready signal to indicate dart coordinates are ready
  output data_taken;            //set to 1 after display has put dart coordinate values into
registers

  output phsync;        // dartboard's horizontal sync
  output pvsync;        // dartboard's vertical sync
  output pblank;        // dartboard's blanking
  output [7:0] pixel;   // dartboard's pixel
  output signed [9:0] x1, x2, x3, y1, y2, y3;
```

```
	parameter screen_width = 1023;		//screen width
	parameter screen_height = 767;			//screen height
	parameter dartboard_height = 453;	//dartboard image height
	parameter dartboard_width = 456;	//dartboard image width - 456 because 3 bits
of padded data added to bitmap
	parameter center_image = 261;			//center dartboard image by this
many pixels from left
	parameter centerxy = 226;		//center of dartboard image
	parameter imagearea = 206568;			//total number of pixels in dartboard
image 453x456

	 reg signed [9:0] x1, x2, x3, y1, y2, y3;	//dart coordinate registers
	wire data_ready;
	 reg data_taken;


	reg [7:0] pixel = 8'b11111111;			//default pixel values
	reg [7:0] pixel_delay = 8'b11111111;
	reg [7:0] pixel_delay2 = 8'b11111111;

	wire [7:0] dbout;			//wire from dartboard image ROM
	 reg [17:0] addr = 0;			//address for dartboard image ROM

	//reg signed[9:0] x1 = 15;		//Test dart data used before integration with
koosh
	//reg signed[9:0] y1 = 55;
	//reg signed[9:0] x2 = 100;
	//reg signed[9:0] y2 = 100;
	//reg signed[9:0] x3 = -100;
	//reg signed[9:0] y3 = 100;

	wire [63:0] score1temp, score2temp, score3temp;  //temporary scores
	wire rdy1, rdy2, rdy3;
	wire [2:0] tpixel7, tpixel8, tpixel9;	//individual dart score text pixels
	parameter[10:0] cx7 = 200;		//locations of individual dart scores below
board
	parameter[9:0]  cy7 = 500;
	parameter[10:0] cx8 = 350;
	parameter[9:0]  cy8 = 500;
	parameter[10:0] cx9 = 500;
	parameter[9:0]  cy9 = 500;

	wire[2:0] dart1, dart2, dart3;

	parameter[10:0] cx1 = 40;
```

```
parameter[9:0]  cy1 = 10;                //PLAYER 1
parameter[10:0] cx2 = 20;
parameter[9:0]  cy2 = 40;                //301
parameter[10:0] cx3 = 800;
parameter[9:0]  cy3 = 10;                //PLAYER 2
parameter[10:0] cx4 = 780;
parameter[9:0]  cy4 = 40;                //301
parameter[10:0] cx5 = 40;
parameter[9:0]  cy5 = 50;                //UNDERLINES
parameter[10:0] cx6 = 800;
parameter[9:0]  cy6 = 50;                //UNDERLINES

wire[2:0] tpixel1;                       //text pixels for game text
wire[2:0] tpixel2;
wire[2:0] tpixel3;
wire[2:0] tpixel4;
wire[2:0] tpixel5;
wire[2:0] tpixel6;

wire [63:0] cstring1 = "PLAYER 1";
reg [23:0] cstring2 = "301";
wire [63:0] cstring3 = "PLAYER 2";
wire [63:0] cstring4 = "________";

reg prev_vsync = 1;    //used to determine falling edge of vsync
reg old_enter = 0;     //used to determine rising edge of enter button

//301 GAME
reg [63:0] score1, score2, score3;  //ascii score of individual darts
wire [31:0] netscore1, netscore2;  //ascii score of player1 and player2
parameter[10:0] gamex1 = 20;                     //P1 SCORE location
parameter[9:0]  gamey1 = 80;
parameter[10:0] gamex2 = 780;                    //P2 SCORE location
parameter[9:0]  gamey2 = 80;
wire[2:0] tpixelp1s, tpixelp2s;
wire tempdouble1, tempdouble2, tempdouble3;
reg double1, double2, double3;
reg ce = 1;                              //clock enable for dartscore
reg [7:0] dbpic_latch1, dbpic_latch2;
wire turn;                               //0-player1, 1- player2
wire turnpixel;
reg [9:0] turnx = 20;                    //turnblob coordinates
reg [10:0] turny = 40;

always @ (posedge vclock) begin
```

```
		old_enter <= enter;		//store previous enter button press to determing rising
edge
		prev_vsync <= vsync; //store previous sync - used to determine falling edge of
vsync
		ce <= 0;

		pixel <= dbpic_latch2;		//Attempt to eliminate dbpicture glitching by
latching image pixel
		dbpic_latch2 <= dbpic_latch1;

		data_taken <= 0;

		if (gamechoice)			//determine 301 vs. 601 on display based on
switch[0]
			cstring2 <= "601";
		else if (!gamechoice)
			cstring2 <= "301";



		if (reset) begin
				addr <= 0;
				ce <= 1;
				turnx <= 20;
				turny <= 40;
		end

		//DISPLAY DARTBOARD, DARTS, AND TEXT
		else if (vcount < dartboard_height && hcount > center_image && hcount <
dartboard_width + center_image) begin
						if (dart1 > 0 || dart2 > 0 || dart3 > 0) begin
							dbpic_latch1 <= 8'b00100111;	//display orange
darts
						end
						else begin
							dbpic_latch1 <= dbout;		//display
dartboard image
						end
				//adjust address to compensate for different arrangement of pixels
in memory
				addr <= vcount*dartboard_width + (dartboard_width-1 +
center_image - hcount);
		end
		else  begin
			dbpic_latch1 <= 8'b11111100 | tpixel1 | tpixel2 | tpixel3 | tpixel4
|tpixel5 | tpixel6 | tpixel7 | tpixel8 | tpixel9 | tpixelp1s | tpixelp2s | turnpixel;
```

```
			if (addr >= imagearea-1)
					addr <= 0;
		end

		//CONVERT KOOSH's DATA POINTS to my coordinate system 0,0 at top left
corner as opposed to bottom left
		if (data_ready) begin
				x1 <= {kooshx1[8], kooshx1};
				x2 <= {kooshx2[8], kooshx2};
				x3 <= {kooshx3[8], kooshx3};
				y1 <= ~{kooshy1[8], kooshy1} + 1;
				y2 <= ~{kooshy2[8], kooshy2} + 1;
				y3 <= ~{kooshy3[8], kooshy3} + 1;
				data_taken <= 1;
		end
		//COUNT UP SCORES FROM INDIVIDUAL DARTS AND SUBMIT TO
GAME LOGIC
		else if (enter && !old_enter) begin
				score1 <= score1temp;
				score2 <= score2temp;
				score3 <= score3temp;
				double1 <= tempdouble1;
				double2 <= tempdouble2;
				double3 <= tempdouble3;
				if (!turn) begin
					turnx <= 770;
					turny <= 40;
				end
				else if (turn) begin
				turnx <= 20;
					turny <= 40;
				end
		 end
		 //CORRECT MISPLACED DARTS
		 else if (!vsync && prev_vsync && dartcorrection == 3'b100) begin
			ce <= 1;
			if (up)
					y1 <= y1 - 1;
			else if (down)
					y1 <= y1 + 1;
			else if (left)
					x1 <= x1 - 1;
			else if (right)
					x1 <= x1 + 1;
		   end
		   else if (!vsync && prev_vsync && dartcorrection == 3'b010)
```

```
                begin
                   ce <= 1;
                   if (up)
                                  y2 <= y2 - 1;
                   else if (down)
                                  y2 <= y2 + 1;
                   else if (left)
                                  x2 <= x2 - 1;
                   else if (right)
                                  x2 <= x2 + 1;
                end
                else if (!vsync && prev_vsync && dartcorrection == 3'b001)
                begin
                   ce <= 1;
                   if (up)
                                  y3 <= y3 - 1;
                   else if (down)
                                  y3 <= y3 + 1;
                   else if (left)
                                  x3 <= x3 - 1;
                   else if (right)
                                  x3 <= x3 + 1;
                end

        end


assign phsync = hsync;
assign pvsync = vsync;
assign pblank = blank;

colordb image_reader(imagearea-1 - addr, vclock, dbout);

//dart display modules
dartblob d1(x1+centerxy,y1+centerxy,hcount-center_image,vcount,dart1)
dartblob d2(x2+centerxy,y2+centerxy,hcount-center_image,vcount,dart2)
dartblob d3(x3+centerxy,y3+centerxy,hcount-center_image,vcount,dart3);

//text display modules
char_string_display text1(vclock,hcount,vcount,tpixel1,cstring1,cx1,cy1);
char_string_display text2(vclock,hcount,vcount,tpixel2,cstring2,cx2,cy2);
char_string_display text3(vclock,hcount,vcount,tpixel3,cstring3,cx3,cy3);
char_string_display text4(vclock,hcount,vcount,tpixel4,cstring2,cx4,cy4);
char_string_display text5(vclock,hcount,vcount,tpixel5,cstring4,cx5,cy5);
char_string_display text6(vclock,hcount,vcount,tpixel6,cstring4,cx6,cy6);
char_string_display scoretext1(vclock,hcount,vcount,tpixel7,score1temp,cx7,cy7);
```

```
   char_string_display scoretext2(vclock,hcount,vcount,tpixel8,score2temp,cx8,cy8);
   char_string_display scoretext3(vclock,hcount,vcount,tpixel9,score3temp,cx9,cy9);

   //returns scores of individual darts
   dartscore dartscore1 (vclock, ce, x1, y1, rdy1, score1temp, tempdouble1);
   dartscore dartscore2 (vclock, ce, x2, y2, rdy2, score2temp, tempdouble2);
   dartscore dartscore3 (vclock, ce, x3, y3, rdy3, score3temp, tempdouble3);


   //301 GAME

   //player 1 and player 2 scores
   char_string_display
player1score(vclock,hcount,vcount,tpixelp1s,netscore1,gamex1,gamey1);
   char_string_display
player2score(vclock,hcount,vcount,tpixelp2s,netscore2,gamex2,gamey2);

   game301 game1(vclock, reset, score1, score2, score3, double1, double2, double3,
netscore1, netscore2, enter, turn, gamechoice);

   turnblob turn_indicator(turnx,turny,hcount,vcount,turnpixel);


endmodule

////////////////////////////////////////////////////////////////////////
//
// Dart Blob:  Places a dart at an x and y coordinate on screen
//
////////////////////////////////////////////////////////////////////////
module dartblob(x,y,hcount,vcount,pixel);
        parameter WIDTH = 4; // default width: 8 pixels...4 pixels to center
        parameter HEIGHT = 4; // default height: 8 pixels
        parameter COLOR = 3'b111; // default color: blue
        input signed[9:0] x;
        input [10:0] hcount;
        input signed[9:0] y;
        input [9:0] vcount;
        output [2:0] pixel;
        reg [2:0] pixel;

        always @ (x or y or hcount or vcount) begin
                if ((hcount >= x-WIDTH && hcount < (x+WIDTH)) &&
                        (vcount >= y-HEIGHT && vcount < (y+HEIGHT)))
                        pixel = COLOR;
                else pixel = 0;
```

```
	end
endmodule
//////////////////////////////////////////////////////////////////////
//
// Turn Blob:  Places a turn indicator at an x and y coordinate
//	turn indicator shows if it is player1 turn or player2 turn
//
//////////////////////////////////////////////////////////////////////
module turnblob(x,y,hcount,vcount,pixel);
	parameter WIDTH = 16; // default width: 8 pixels
	parameter HEIGHT = 16; // default height: 8 pixels
	parameter COLOR = 3'b111; // default color: red
	input [9:0] x;
	input [10:0] hcount;
	input [9:0] y;
	input [9:0] vcount;
	output [2:0] pixel;
	reg [2:0] pixel;

	always @ (x or y or hcount or vcount) begin
		if ((hcount >= x-WIDTH && hcount < (x+WIDTH)) &&
			(vcount >= y-HEIGHT && vcount < (y+HEIGHT)))
			pixel = COLOR;
		else pixel = 0;
	end
endmodule

//////////////////////////////////////////////////////////////////////
//
// Score Module:  Converts x-y coordinates to polar coordinates and returns ascii value
//
//////////////////////////////////////////////////////////////////////
module dartscore(vclock, ce, x, y, rdy, score, double);
	input vclock;			//65 MHz clock
	input signed [9:0] x;			//Between -230 and 230
	input signed [9:0] y;			//Between -230 and 230
	input ce;				//clock enabled if 1
	output rdy;			//1 - output read
	output [63:0] score;			//ascii score between 0 and 60.  uses low
order bits unless "NO SCORE"
	output double;			//1 - in double scoring region
	wire signed [9:0] theta, r;		//polar coordinates between -1 and 1 in
2QN decimal format
	wire signed [9:0] x, y;
	reg signed [9:0] invtheta;		//2's complement of theta
	reg [63:0] score;			//ascii score
```

```
        reg double;
        wire rdy;

        //parameters indicate score boundaries on actual dartboard as decimals
        //output theta between -1 and 1 so each boundary is .1 wide
        //format is high order bit sign, second and third high order bits integers all others
decimal

        parameter [9:0]theta0 = 10'b0000000000;  //0.0
        parameter [9:0]theta5 = 10'b0000000110;  //.05
        parameter [9:0]theta15 = 10'b0000010100;  //.15
        parameter [9:0]theta25 = 10'b0000100000;  //.25
        parameter [9:0]theta35 = 10'b0000101100;  //.35
        parameter [9:0]theta45 = 10'b0000111010;  //.45
        parameter [9:0]theta55 = 10'b0001000110;  //.55
        parameter [9:0]theta65 = 10'b0001010100;  //.65
        parameter [9:0]theta75 = 10'b0001100000;  //.75
        parameter [9:0]theta85 = 10'b0001101100;  //.85
        parameter [9:0]theta95 = 10'b0001111010;  //.95
        parameter [9:0]theta1  = 10'b0010000000;  //1.0


        always @ (posedge vclock) begin

                if (ce) begin

                //ORDER OF IF STATEMENTS STARTS WITH 0 DEGREES AND
CONTINUES CCW
                if (r<=170) begin  //if dart is on the board
                        double <= 0;
                        invtheta <= ~theta + 1;
                        if (r <= 7) begin  //double bullseye
                                score <= "50";
                                double <= 1;
                        end
                        else if (r > 7 && r <= 16)  //single bullseye
                                score <= "25";
                        else if (theta > theta0 && theta < theta5) begin         //Bottom
Half Of Board
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "18";
                                else if (r >= 162 && r<= 170)           begin //double
                                        score <= "12";
                                        double <= 1;
                                end
                                else                                            //single
```

```
                                        score <= "06";
                                end
                        else if (theta >= theta5  && theta < theta15)begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "30";
                                else if (r >= 162 && r<= 170) begin   //double
                                        score <= "20";
                                        double <= 1;
                                end
                                else                                        //single
                                        score <= "10";
                                end
                        else if (theta >= theta15 && theta < theta25)begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "45";
                                else if (r >= 162 && r<= 170)           begin //double
                                        score <= "30";
                                        double <= 1;
                                end
                                else                                        //single
                                        score <= "15";
                                end
                        else if (theta >= theta25 && theta < theta35) begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "06";
                                else if (r >= 162 && r<= 170)           begin
//double
                                        score <= "04";
                                        double <= 1;
                                end
                                else                                        //single
                                        score <= "02";
                                end
                        else if (theta >= theta35 && theta < theta45)
                                begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "51";
                                else if (r >= 162 && r<= 170) begin //double
                                        score <= "34";
                                        double <= 1;
                                end
                                else                                        //single
                                        score <= "17";
                                end
                        else if (theta >= theta45 && theta < theta55)begin
                                if (r >= 99 && r<= 107)         //triple
```

```
                score <= "09";
        else if (r >= 162 && r<= 170)begin //double
                score <= "06";
                double <= 1;
        end
        else                                            //single
                score <= "03";
        end
else if (theta >= theta55 && theta < theta65)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "57";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "38";
                double <= 1;
        end
        else                                            //single
                score <= "19";
        end
else if (theta >= theta65 && theta < theta75)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "21";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "14";
                double <= 1;
        end
        else                                            //single
                score <= "07";
        end
else if (theta >= theta75 && theta < theta85)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "48";
        else if (r >= 162 && r<= 170)begin //double
                score <= "32";
                double <= 1;
        end
        else                                            //single
                score <= "16";
        end
else if (theta >= theta85 && theta < theta95)
begin
        if (r >= 99 && r<= 107)         //triple
                score <= "24";
        else if (r >= 162 && r<= 170)           begin //double
```

```
                                        score <= "16";
                                        double <= 1;
                                end
                                else                                            //single
                                        score <= "08";
                                end
                        else if (theta >= theta95 && theta < theta1)
                                begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "33";
                                else if (r >= 162 && r<= 170)           begin //double
                                        score <= "22";
                                        double <= 1;
                                end
                                else                                            //single
                                        score <= "11";
                                end

                        else if (invtheta >= theta0 && invtheta < theta5)       //Top Half Of
Board
                                begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "18";
                                else if (r >= 162 && r<= 170)           begin //double
                                        score <= "12";
                                        double <= 1;
                                end
                                else                                            //single
                                        score <= "06";
                                end
                else if (invtheta >= theta5  && invtheta < theta15)
                                begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "39";
                                else if (r >= 162 && r<= 170)          begin //double
                                        score <= "26";
                                        double <= 1;
                                end
                                else                                            //single
                                        score <= "13";
                                end
                        else if (invtheta >= theta15 && invtheta < theta25)
                                begin
                                if (r >= 99 && r<= 107)         //triple
                                        score <= "12";
                                else if (r >= 162 && r<= 170)           begin //double
                score <= "08";
                double <= 1;
            end
            else                                            //single
                score <= "04";
            end
        else if (invtheta >= theta25 && invtheta < theta35)
            begin
            if (r >= 99 && r<= 107)          //triple
                score <= "54";
            else if (r >= 162 && r<= 170)           begin //double
                score <= "36";
                double <= 1;
            end
            else                                            //single
                score <= "18";
            end
        else if (invtheta >= theta35 && invtheta < theta45)
            begin
            if (r >= 99 && r<= 107)          //triple
                score <= "03";
            else if (r >= 162 && r<= 170)           begin //double
                score <= "02";
                double <= 1;
            end
            else                                            //single
                score <= "01";
            end
        else if (invtheta >= theta45 && invtheta < theta55)
            begin
            if (r >= 99 && r<= 107)          //triple
                score <= "60";
            else if (r >= 162 && r<= 170)           begin //double
                score <= "40";
                double <= 1;
            end
            else                                            //single
                score <= "20";
            end
        else if (invtheta >= theta55 && invtheta < theta65)
            begin
            if (r >= 99 && r<= 107)          //triple
                score <= "15";
            else if (r >= 162 && r<= 170)           begin //double
                score <= "10";
                double <= 1;
        end
        else                                        //single
                score <= "05";
        end
else if (invtheta >= theta65 && invtheta < theta75)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "36";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "24";
                double <= 1;
        end
        else                                        //single
                score <= "12";
        end
else if (invtheta >= theta75 && invtheta < theta85)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "27";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "18";
                double <= 1;
        end
        else                                        //single
                score <= "09";
        end
else if (invtheta >= theta85 && invtheta < theta95)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "42";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "28";
                double <= 1;
        end
        else                                        //single
                score <= "14";
        end
else if (invtheta >= theta95 && invtheta <= theta1)
        begin
        if (r >= 99 && r<= 107)         //triple
                score <= "33";
        else if (r >= 162 && r<= 170)           begin //double
                score <= "22";
                double <= 1;
        end
        else                                        //single
```

```
                                        score <= "11";
                                end
                        else
                                score <= "NO SCORE";
                end
                else
                        score <= "NO SCORE";
        end
        end
        polargen convert1(x, y, vclock, ce, r, theta, rdy);  //converts x,y to r, theta

endmodule


///////////////////////////////////////////////////////////////////
//
// game301:  Maintains state of 301 game.  Enforces
//                double in and double out rules.  Outputs
//                ascii score of player1 and player2 in addition
//                to whose turn it currently is.
//
///////////////////////////////////////////////////////////////////
module game301(vclock, reset, score1, score2, score3, double1, double2, double3,
netscore1, netscore2, enter, turn, gamechoice);

        input vclock;                   //65 MHZ clock
        input reset;                    //reset
        input [63:0] score1, score2, score3;    //input dart scores
        input double1, double2, double3;              //double indicators- 1 in double region
0 not in double region
        input enter;                    //indicates when to update scores
        input gamechoice;                       //0: 301, 1:601
        output[31:0] netscore1;                  //player1 ascii net score
        output[31:0] netscore2;                 //player1 ascii net score
        output turn;                    //0 - player1 turn      1- player2 turn

        reg [9:0] score = 0;
        reg [9:0] playerscore1 = 301;
        reg [9:0] playerscore2 = 301;
        reg scoreschanged = 0;
        reg [9:0] maxscore = 301;
        reg turn = 0;
        reg player1win = 0;
        reg player2win = 0;
        reg bust1 = 0;
        reg bust2 = 0;
```

```
	reg logic = 0;
	reg old_enter = 0;
	reg old_enter2 = 0;
	wire [23:0] dout1;
	wire [23:0] dout2;

	always @ (posedge vclock) begin
	  old_enter2 <= enter;
	  old_enter <= old_enter2;		//Need 2 due to two clock cycle delay from
main module

	  if (gamechoice && !scoreschanged)begin
			playerscore1 <= 601;
			playerscore2 <= 601;
			maxscore <= 601;
	  end
	  else if (!gamechoice && !scoreschanged)begin
			playerscore1 <= 301;
			playerscore2 <= 301;
			maxscore <= 301;
	  end

	  if (reset)	begin
			if (!gamechoice)	begin
				playerscore1 <= 301;
				playerscore2 <= 301;
				maxscore <= 301;
			end
			else if (gamechoice)	begin
				playerscore1 <= 601;
				playerscore2 <= 601;
				maxscore <= 601;
			end

			turn <= 0;
			score <= 0;
			player1win <= 0;
			player2win <= 0;
			bust1 <= 0;
			bust2 <= 0;
			logic <= 0;
			old_enter <=0;
			old_enter2 <= 0;
			scoreschanged <= 0;
	  end
```

```
            else if (old_enter2 && !old_enter && ((!turn && playerscore1 == maxscore) ||
(turn && playerscore2 == maxscore)))        begin    //PLAYER1 NEEDS TO DOUBLE
IN
                              logic <= 1;
                              if (score1 == "NO SCORE" && score2 == "NO SCORE" &&
score3 == "NO SCORE")
                                 score <= 0;
                              else if (!double1 && !double2 && !double3)
                                 score <= 0;
                              else     if (score1 == "NO SCORE" && score2 == "NO SCORE"
&& double3)
                                 score <= score3[7:0] - 48 + 10*(score3[15:8] - 48);
                              else     if (score2 == "NO SCORE" && score3 == "NO SCORE"
&& double1)
                                 score <= score1[7:0] - 48 + 10*(score1[15:8] - 48);
                              else     if (score1 == "NO SCORE" && score3 == "NO SCORE"
&& double2)
                                 score <= score2[7:0] - 48 + 10*(score2[15:8] - 48);
                              else     if (score1 == "NO SCORE" && double2)
                                 score <= score2[7:0] - 48 + 10*(score2[15:8] - 48) + score3[7:0]
- 48 + 10*(score3[15:8] - 48);
                              else     if (score1 == "NO SCORE" && double3)
                                 score <= score3[7:0] - 48 + 10*(score3[15:8] - 48);
                              else     if (score2 == "NO SCORE" && double1)
                                 score <= score1[7:0] - 48 + 10*(score1[15:8] - 48) + score3[7:0]
- 48 + 10*(score3[15:8] - 48);
                              else     if (score2 == "NO SCORE" && double3)
                                 score <= score3[7:0] - 48 + 10*(score3[15:8] - 48);

                              else     if (score3 == "NO SCORE" && double1)
                                 score <= score1[7:0] - 48 + 10*(score1[15:8] - 48) + score2[7:0]
- 48 + 10*(score2[15:8] - 48);
                              else     if (score3 == "NO SCORE" && double2)
                                 score <= score2[7:0] - 48 + 10*(score2[15:8] - 48);

                              else     if (double1)
                                 score <= score1[7:0] - 48 + 10*(score1[15:8] - 48)+ score2[7:0]
- 48 + 10*(score2[15:8] - 48) + score3[7:0] - 48 + 10*(score3[15:8] - 48);
                              else if (double2)
                                 score <= score2[7:0] - 48 + 10*(score2[15:8] - 48) + score3[7:0]
- 48 + 10*(score3[15:8] - 48);
                              else if (double3)
                                 score <= score3[7:0] - 48 + 10*(score3[15:8] - 48);
                              else
                                 score <= 0;
                  end
            //IF PLAYER HAS ALREADY DOUBLED IN
            else if (old_enter2 && !old_enter && ((!turn && playerscore1 <
maxscore)  || (turn && playerscore2 < maxscore))) begin
                  logic <= 1;
                  if (score1 == "NO SCORE" && score2 == "NO SCORE" &&
score3 == "NO SCORE")
                    score <= 0;
                  else    if (score1 == "NO SCORE" && score2 == "NO SCORE")
                    score <= score3[7:0] - 48 + 10*(score3[15:8] - 48);
                  else    if (score2 == "NO SCORE" && score3 == "NO SCORE")
                    score <= score1[7:0] - 48 + 10*(score1[15:8] - 48);
                  else    if (score1 == "NO SCORE" && score3 == "NO SCORE")
                    score <= score2[7:0] - 48 + 10*(score2[15:8] - 48);
                  else    if (score1 == "NO SCORE")
                    score <= score2[7:0] - 48 + 10*(score2[15:8] - 48) + score3[7:0]
- 48 + 10*(score3[15:8] - 48);
                  else    if (score2 == "NO SCORE")
                    score <= score1[7:0] - 48 + 10*(score1[15:8] - 48) + score3[7:0]
- 48 + 10*(score3[15:8] - 48);
                  else    if (score3 == "NO SCORE")
                    score <= score1[7:0] - 48 + 10*(score1[15:8] - 48) + score2[7:0]
- 48 + 10*(score2[15:8] - 48);
                  else
                    score <= score1[7:0] - 48 + 10*(score1[15:8] - 48)+ score2[7:0]
- 48 + 10*(score2[15:8] - 48) + score3[7:0] - 48 + 10*(score3[15:8] - 48);

            end
        //UPDATE SCORES
         if (logic && !player1win && !player2win) begin
                  logic <= 0;
                  scoreschanged <= 1;
                  if (!turn && score < playerscore1 - 1) begin              // - 1
because a 1 is a bust
                        playerscore1 <= playerscore1 - score;
                        bust1 <= 0;
                        bust2 <= 0;
                        turn <= turn + 1;
                  end
                  else if (!turn && score == playerscore1 && ((double1 && score2
== "NO SCORE" && score3 == "NO SCORE") || (double2 && score3 == "NO
SCORE") || double3)) begin
                        playerscore1 <= 0;
                        player1win <= 1;
                        bust1 <= 0;
                        bust2 <= 0;
                        turn <= turn + 1;
```

```
				end
				else if (!turn && (score > playerscore1 - 1 || !(score == 
playerscore1 && ((double1 && score2 == "NO SCORE" && score3 == "NO SCORE") 
|| (double2 && score3 == "NO SCORE") || double3)))) begin				// - 1 
because a 1 is a bust
					bust1 <= 1;
					bust2 <= 0;
					turn <= turn + 1;
				end
				else if (turn && score < playerscore2 - 1)begin
					playerscore2 <= playerscore2 - score;
					bust1 <= 0;
					bust2 <= 0;
					turn <= turn + 1;
				end
				else if (turn && score == playerscore2 && ((double1 && score2 
== "NO SCORE" && score3 == "NO SCORE") || (double2 && score3 == "NO 
SCORE") || double3)) begin
					playerscore2 <= 0;
					player2win <= 1;
					bust1 <= 0;
					bust2 <= 0;
					turn <= turn + 1;
				end
				else if (turn && (score > playerscore2 - 1 || !(score == 
playerscore2 && ((double1 && score2 == "NO SCORE" && score3 == "NO SCORE") 
|| (double2 && score3 == "NO SCORE") || double3))))begin
					bust1 <= 0;
					bust2 <= 1;
					turn <= turn + 1;
				end

		  end
		end

		//output netscore or "WIN" or "BUST" if player has gone below zero
		assign netscore1 = (bust1) ? "BUST" : (player1win) ? "WIN" : {8'b00000000, 
dout1};
		assign netscore2 = (bust2) ? "BUST" : (player2win) ? "WIN" : {8'b00000000, 
dout2};

		//LUT converts binary to ascii net score
		numtotext num1(vclock, playerscore1, dout1);
		numtotext num2(vclock, playerscore2, dout2);
```

```
endmodule




/////////////////////////////////////////////////////////////////////
//
// Convert  8 bit bitmap color palette LUT
//
//
/////////////////////////////////////////////////////////////////////
module convertcolor(vclock, rgb, r, g, b);
input vclock;
input [7:0] rgb;
output [7:0] r, g, b;

reg [7:0] r, g, b;

  always @ (posedge vclock) begin
	case(rgb)
	  0: begin b <= 0; g <= 0; r <= 0; end
	   1: begin b <= 0; g <= 0; r <= 128;end
	   2: begin b <= 0; g <= 128; r <= 0;end
	   3: begin b <= 0; g <= 128; r <= 128;end
	   4: begin b <= 128; g <= 0; r <= 0;end
	   5: begin b <= 128; g <= 0; r <= 128;end
	   6: begin b <= 128; g <= 128; r <= 0;end
	   7: begin b <= 192; g <= 192; r <= 192;end
	   8: begin b <= 192; g <= 220; r <= 192;end
	   9: begin b <= 240; g <= 202; r <= 166;end
	   10: begin b <= 0; g <= 32; r <= 64;end
	   11: begin b <= 0; g <= 32; r <= 96;end
	   12: begin b <= 0; g <= 32; r <= 128;end
	   13: begin b <= 0; g <= 32; r <= 160;end
	   14: begin b <= 0; g <= 32; r <= 192;end
	   15: begin b <= 0; g <= 32; r <= 224;end
	   16: begin b <= 0; g <= 64; r <= 0;end
	   17: begin b <= 0; g <= 64; r <= 32;end
	   18: begin b <= 0; g <= 64; r <= 64;end
	   19: begin b <= 0; g <= 64; r <= 96;end
	   20: begin b <= 0; g <= 64; r <= 128;end
	   21: begin b <= 0; g <= 64; r <= 160;end
	   22: begin b <= 0; g <= 64; r <= 192;end
	   23: begin b <= 0; g <= 64; r <= 224;end
```

```
24: begin b <= 0; g <= 96; r <= 0;end
25: begin b <= 0; g <= 96; r <= 32;end
26: begin b <= 0; g <= 96; r <= 64;end
27: begin b <= 0; g <= 96; r <= 96;end
28: begin b <= 0; g <= 96; r <= 128;end
29: begin b <= 0; g <= 96; r <= 160;end
30: begin b <= 0; g <= 96; r <= 192;end
31: begin b <= 0; g <= 96; r <= 224;end
32: begin b <= 0; g <= 128; r <= 0;end
33: begin b <= 0; g <= 128; r <= 32;end
34: begin b <= 0; g <= 128; r <= 64;end
35: begin b <= 0; g <= 128; r <= 96;end
36: begin b <= 0; g <= 128; r <= 128;end
37: begin b <= 0; g <= 128; r <= 160;end
38: begin b <= 0; g <= 128; r <= 192;end
39: begin b <= 0; g <= 128; r <= 224;end
40: begin b <= 0; g <= 160; r <= 0;end
41: begin b <= 0; g <= 160; r <= 32;end
42: begin b <= 0; g <= 160; r <= 64;end
43: begin b <= 0; g <= 160; r <= 96;end
44: begin b <= 0; g <= 160; r <= 128;end
45: begin b <= 0; g <= 160; r <= 160;end
46: begin b <= 0; g <= 160; r <= 192;end
47: begin b <= 0; g <= 160; r <= 224;end
48: begin b <= 0; g <= 192; r <= 0;end
49: begin b <= 0; g <= 192; r <= 32;end
50: begin b <= 0; g <= 192; r <= 64;end
51: begin b <= 0; g <= 192; r <= 96;end
52: begin b <= 0; g <= 192; r <= 128;end
53: begin b <= 0; g <= 192; r <= 160;end
54: begin b <= 0; g <= 192; r <= 192;end
55: begin b <= 0; g <= 192; r <= 224;end
56: begin b <= 0; g <= 224; r <= 0;end
57: begin b <= 0; g <= 224; r <= 32;end
58: begin b <= 0; g <= 224; r <= 64;end
59: begin b <= 0; g <= 224; r <= 96;end
60: begin b <= 0; g <= 224; r <= 128;end
61: begin b <= 0; g <= 224; r <= 160;end
62: begin b <= 0; g <= 224; r <= 192;end
63: begin b <= 0; g <= 224; r <= 224;end
64: begin b <= 64; g <= 0; r <= 0;end
65: begin b <= 64; g <= 0; r <= 32;end
66: begin b <= 64; g <= 0; r <= 64;end
67: begin b <= 64; g <= 0; r <= 96;end
68: begin b <= 64; g <= 0; r <= 128;end
69: begin b <= 64; g <= 0; r <= 160;end
```

```
70: begin b <= 64; g <= 0; r <= 192;end
71: begin b <= 64; g <= 0; r <= 224;end
72: begin b <= 64; g <= 32; r <= 0;end
73: begin b <= 64; g <= 32; r <= 32;end
74: begin b <= 64; g <= 32; r <= 64;end
75: begin b <= 64; g <= 32; r <= 96;end
76: begin b <= 64; g <= 32; r <= 128;end
77: begin b <= 64; g <= 32; r <= 160;end
78: begin b <= 64; g <= 32; r <= 192;end
79: begin b <= 64; g <= 32; r <= 224;end
80: begin b <= 64; g <= 64; r <= 0;end
81: begin b <= 64; g <= 64; r <= 32;end
82: begin b <= 64; g <= 64; r <= 64;end
83: begin b <= 64; g <= 64; r <= 96;end
84: begin b <= 64; g <= 64; r <= 128;end
85: begin b <= 64; g <= 64; r <= 160;end
86: begin b <= 64; g <= 64; r <= 192;end
87: begin b <= 64; g <= 64; r <= 224;end
88: begin b <= 64; g <= 96; r <= 0;end
89: begin b <= 64; g <= 96; r <= 32;end
90: begin b <= 64; g <= 96; r <= 64;end
91: begin b <= 64; g <= 96; r <= 96;end
92: begin b <= 64; g <= 96; r <= 128;end
93: begin b <= 64; g <= 96; r <= 160;end
94: begin b <= 64; g <= 96; r <= 192;end
95: begin b <= 64; g <= 96; r <= 224;end
96: begin b <= 64; g <= 128; r <= 0;end
97: begin b <= 64; g <= 128; r <= 32;end
98: begin b <= 64; g <= 128; r <= 64;end
99: begin b <= 64; g <= 128; r <= 96;end
100: begin b <= 64; g <= 128; r <= 128;end
101: begin b <= 64; g <= 128; r <= 160;end
102: begin b <= 64; g <= 128; r <= 192;end
103: begin b <= 64; g <= 128; r <= 224;end
104: begin b <= 64; g <= 160; r <= 0;end
105: begin b <= 64; g <= 160; r <= 32;end
106: begin b <= 64; g <= 160; r <= 64;end
107: begin b <= 64; g <= 160; r <= 96;end
108: begin b <= 64; g <= 160; r <= 128;end
109: begin b <= 64; g <= 160; r <= 160;end
110: begin b <= 64; g <= 160; r <= 192;end
111: begin b <= 64; g <= 160; r <= 224;end
112: begin b <= 64; g <= 192; r <= 0;end
113: begin b <= 64; g <= 192; r <= 32;end
114: begin b <= 64; g <= 192; r <= 64;end
115: begin b <= 64; g <= 192; r <= 96;end
```

```
116: begin b <= 64; g <= 192; r <= 128;end
117: begin b <= 64; g <= 192; r <= 160;end
118: begin b <= 64; g <= 192; r <= 192;end
119: begin b <= 64; g <= 192; r <= 224;end
120: begin b <= 64; g <= 224; r <= 0;end
121: begin b <= 64; g <= 224; r <= 32;end
122: begin b <= 64; g <= 224; r <= 64;end
123: begin b <= 64; g <= 224; r <= 96;end
124: begin b <= 64; g <= 224; r <= 128;end
125: begin b <= 64; g <= 224; r <= 160;end
126: begin b <= 64; g <= 224; r <= 192;end
127: begin b <= 64; g <= 224; r <= 224;end
128: begin b <= 128; g <= 0; r <= 0;end
129: begin b <= 128; g <= 0; r <= 32;end
130: begin b <= 128; g <= 0; r <= 64;end
131: begin b <= 128; g <= 0; r <= 96;end
132: begin b <= 128; g <= 0; r <= 128;end
133: begin b <= 128; g <= 0; r <= 160;end
134: begin b <= 128; g <= 0; r <= 192;end
135: begin b <= 128; g <= 0; r <= 224;end
136: begin b <= 128; g <= 32; r <= 0;end
137: begin b <= 128; g <= 32; r <= 32;end
138: begin b <= 128; g <= 32; r <= 64;end
139: begin b <= 128; g <= 32; r <= 96;end
140: begin b <= 128; g <= 32; r <= 128;end
141: begin b <= 128; g <= 32; r <= 160;end
142: begin b <= 128; g <= 32; r <= 192;end
143: begin b <= 128; g <= 32; r <= 224;end
144: begin b <= 128; g <= 64; r <= 0;end
145: begin b <= 128; g <= 64; r <= 32;end
146: begin b <= 128; g <= 64; r <= 64;end
147: begin b <= 128; g <= 64; r <= 96;end
148: begin b <= 128; g <= 64; r <= 128;end
149: begin b <= 128; g <= 64; r <= 160;end
150: begin b <= 128; g <= 64; r <= 192;end
151: begin b <= 128; g <= 64; r <= 224;end
152: begin b <= 128; g <= 96; r <= 0;end
153: begin b <= 128; g <= 96; r <= 32;end
154: begin b <= 128; g <= 96; r <= 64;end
155: begin b <= 128; g <= 96; r <= 96;end
156: begin b <= 128; g <= 96; r <= 128;end
157: begin b <= 128; g <= 96; r <= 160;end
158: begin b <= 128; g <= 96; r <= 192;end
159: begin b <= 128; g <= 96; r <= 224;end
160: begin b <= 128; g <= 128; r <= 0;end
161: begin b <= 128; g <= 128; r <= 32;end
```

```
162: begin b <= 128; g <= 128; r <= 64;end
163: begin b <= 128; g <= 128; r <= 96;end
164: begin b <= 128; g <= 128; r <= 128;end
165: begin b <= 128; g <= 128; r <= 160;end
166: begin b <= 128; g <= 128; r <= 192;end
167: begin b <= 128; g <= 128; r <= 224;end
168: begin b <= 128; g <= 160; r <= 0;end
169: begin b <= 128; g <= 160; r <= 32;end
170: begin b <= 128; g <= 160; r <= 64;end
171: begin b <= 128; g <= 160; r <= 96;end
172: begin b <= 128; g <= 160; r <= 128;end
173: begin b <= 128; g <= 160; r <= 160;end
174: begin b <= 128; g <= 160; r <= 192;end
175: begin b <= 128; g <= 160; r <= 224;end
176: begin b <= 128; g <= 192; r <= 0;end
177: begin b <= 128; g <= 192; r <= 32;end
178: begin b <= 128; g <= 192; r <= 64;end
179: begin b <= 128; g <= 192; r <= 96;end
180: begin b <= 128; g <= 192; r <= 128;end
181: begin b <= 128; g <= 192; r <= 160;end
182: begin b <= 128; g <= 192; r <= 192;end
183: begin b <= 128; g <= 192; r <= 224;end
184: begin b <= 128; g <= 224; r <= 0;end
185: begin b <= 128; g <= 224; r <= 32;end
186: begin b <= 128; g <= 224; r <= 64;end
187: begin b <= 128; g <= 224; r <= 96;end
188: begin b <= 128; g <= 224; r <= 128;end
189: begin b <= 128; g <= 224; r <= 160;end
190: begin b <= 128; g <= 224; r <= 192;end
191: begin b <= 128; g <= 224; r <= 224;end
192: begin b <= 192; g <= 0; r <= 0;end
193: begin b <= 192; g <= 0; r <= 32;end
194: begin b <= 192; g <= 0; r <= 64;end
195: begin b <= 192; g <= 0; r <= 96;end
196: begin b <= 192; g <= 0; r <= 128;end
197: begin b <= 192; g <= 0; r <= 160;end
198: begin b <= 192; g <= 0; r <= 192;end
199: begin b <= 192; g <= 0; r <= 224;end
200: begin b <= 192; g <= 32; r <= 0;end
201: begin b <= 192; g <= 32; r <= 32;end
202: begin b <= 192; g <= 32; r <= 64;end
203: begin b <= 192; g <= 32; r <= 96;end
204: begin b <= 192; g <= 32; r <= 128;end
205: begin b <= 192; g <= 32; r <= 160;end
206: begin b <= 192; g <= 32; r <= 192;end
207: begin b <= 192; g <= 32; r <= 224;end
```

```
208: begin b <= 192; g <= 64; r <= 0;end
209: begin b <= 192; g <= 64; r <= 32;end
210: begin b <= 192; g <= 64; r <= 64;end
211: begin b <= 192; g <= 64; r <= 96;end
212: begin b <= 192; g <= 64; r <= 128;end
213: begin b <= 192; g <= 64; r <= 160;end
214: begin b <= 192; g <= 64; r <= 192;end
215: begin b <= 192; g <= 64; r <= 224;end
216: begin b <= 192; g <= 96; r <= 0;end
217: begin b <= 192; g <= 96; r <= 32;end
218: begin b <= 192; g <= 96; r <= 64;end
219: begin b <= 192; g <= 96; r <= 96;end
220: begin b <= 192; g <= 96; r <= 128;end
221: begin b <= 192; g <= 96; r <= 160;end
222: begin b <= 192; g <= 96; r <= 192;end
223: begin b <= 192; g <= 96; r <= 224;end
224: begin b <= 192; g <= 128; r <= 0;end
225: begin b <= 192; g <= 128; r <= 32;end
226: begin b <= 192; g <= 128; r <= 64;end
227: begin b <= 192; g <= 128; r <= 96;end
228: begin b <= 192; g <= 128; r <= 128;end
229: begin b <= 192; g <= 128; r <= 160;end
230: begin b <= 192; g <= 128; r <= 192;end
231: begin b <= 192; g <= 128; r <= 224;end
232: begin b <= 192; g <= 160; r <= 0;end
233: begin b <= 192; g <= 160; r <= 32;end
234: begin b <= 192; g <= 160; r <= 64;end
235: begin b <= 192; g <= 160; r <= 96;end
236: begin b <= 192; g <= 160; r <= 128;end
237: begin b <= 192; g <= 160; r <= 160;end
238: begin b <= 192; g <= 160; r <= 192;end
239: begin b <= 192; g <= 160; r <= 224;end
240: begin b <= 192; g <= 192; r <= 0;end
241: begin b <= 192; g <= 192; r <= 32;end
242: begin b <= 192; g <= 192; r <= 64;end
243: begin b <= 192; g <= 192; r <= 96;end
244: begin b <= 192; g <= 192; r <= 128;end
245: begin b <= 192; g <= 192; r <= 160;end
246: begin b <= 240; g <= 251; r <= 255;end
247: begin b <= 164; g <= 160; r <= 160;end
248: begin b <= 128; g <= 128; r <= 128;end
249: begin b <= 0; g <= 0; r <= 255;end
250: begin b <= 0; g <= 255; r <= 0;end
251: begin b <= 0; g <= 255; r <= 255;end
252: begin b <= 255; g <= 0; r <= 0;end
253: begin b <= 255; g <= 0; r <= 255;end
```

```
		254: begin b <= 255; g <= 255; r <= 0;end
		255: begin b <= 255; g <= 255; r <= 255;end




	endcase
  end
endmodule
///////////////////////////////////////////////////////////////////
//
// Convert Decimal to Ascii values for text display
//
///////////////////////////////////////////////////////////////////


module numtotext (vclock, score, txtscore);
	input vclock;
	input[9:0] score;
	output[23:0] txtscore;

	reg [23:0] txtscore;

	always @(posedge vclock) begin

		case	(score)
			0:txtscore<="  0";
			1:txtscore<="  1";
			2:txtscore<="  2";
			3:txtscore<="  3";
			4:txtscore<="  4";
			5:txtscore<="  5";
			6:txtscore<="  6";
			7:txtscore<="  7";
			8:txtscore<="  8";
			9:txtscore<="  9";
			10:txtscore<=" 10";
			11:txtscore<=" 11";
			12:txtscore<=" 12";
			13:txtscore<=" 13";
			14:txtscore<=" 14";
			15:txtscore<=" 15";
			16:txtscore<=" 16";
			17:txtscore<=" 17";
			18:txtscore<=" 18";
			19:txtscore<=" 19";
```

```
20:txtscore<=" 20";
21:txtscore<=" 21";
22:txtscore<=" 22";
23:txtscore<=" 23";
24:txtscore<=" 24";
25:txtscore<=" 25";
26:txtscore<=" 26";
27:txtscore<=" 27";
28:txtscore<=" 28";
29:txtscore<=" 29";
30:txtscore<=" 30";
31:txtscore<=" 31";
32:txtscore<=" 32";
33:txtscore<=" 33";
34:txtscore<=" 34";
35:txtscore<=" 35";
36:txtscore<=" 36";
37:txtscore<=" 37";
38:txtscore<=" 38";
39:txtscore<=" 39";
40:txtscore<=" 40";
41:txtscore<=" 41";
42:txtscore<=" 42";
43:txtscore<=" 43";
44:txtscore<=" 44";
45:txtscore<=" 45";
46:txtscore<=" 46";
47:txtscore<=" 47";
48:txtscore<=" 48";
49:txtscore<=" 49";
50:txtscore<=" 50";
51:txtscore<=" 51";
52:txtscore<=" 52";
53:txtscore<=" 53";
54:txtscore<=" 54";
55:txtscore<=" 55";
56:txtscore<=" 56";
57:txtscore<=" 57";
58:txtscore<=" 58";
59:txtscore<=" 59";
60:txtscore<=" 60";
61:txtscore<=" 61";
62:txtscore<=" 62";
63:txtscore<=" 63";
64:txtscore<=" 64";
65:txtscore<=" 65";
```

```
66:txtscore<=" 66";
67:txtscore<=" 67";
68:txtscore<=" 68";
69:txtscore<=" 69";
70:txtscore<=" 70";
71:txtscore<=" 71";
72:txtscore<=" 72";
73:txtscore<=" 73";
74:txtscore<=" 74";
75:txtscore<=" 75";
76:txtscore<=" 76";
77:txtscore<=" 77";
78:txtscore<=" 78";
79:txtscore<=" 79";
80:txtscore<=" 80";
81:txtscore<=" 81";
82:txtscore<=" 82";
83:txtscore<=" 83";
84:txtscore<=" 84";
85:txtscore<=" 85";
86:txtscore<=" 86";
87:txtscore<=" 87";
88:txtscore<=" 88";
89:txtscore<=" 89";
90:txtscore<=" 90";
91:txtscore<=" 91";
92:txtscore<=" 92";
93:txtscore<=" 93";
94:txtscore<=" 94";
95:txtscore<=" 95";
96:txtscore<=" 96";
97:txtscore<=" 97";
98:txtscore<=" 98";
99:txtscore<=" 99";
100:txtscore<="100";
101:txtscore<="101";
102:txtscore<="102";
103:txtscore<="103";
104:txtscore<="104";
105:txtscore<="105";
106:txtscore<="106";
107:txtscore<="107";
108:txtscore<="108";
109:txtscore<="109";
110:txtscore<="110";
111:txtscore<="111";
```

```
112:txtscore<="112";
113:txtscore<="113";
114:txtscore<="114";
115:txtscore<="115";
116:txtscore<="116";
117:txtscore<="117";
118:txtscore<="118";
119:txtscore<="119";
120:txtscore<="120";
121:txtscore<="121";
122:txtscore<="122";
123:txtscore<="123";
124:txtscore<="124";
125:txtscore<="125";
126:txtscore<="126";
127:txtscore<="127";
128:txtscore<="128";
129:txtscore<="129";
130:txtscore<="130";
131:txtscore<="131";
132:txtscore<="132";
133:txtscore<="133";
134:txtscore<="134";
135:txtscore<="135";
136:txtscore<="136";
137:txtscore<="137";
138:txtscore<="138";
139:txtscore<="139";
140:txtscore<="140";
141:txtscore<="141";
142:txtscore<="142";
143:txtscore<="143";
144:txtscore<="144";
145:txtscore<="145";
146:txtscore<="146";
147:txtscore<="147";
148:txtscore<="148";
149:txtscore<="149";
150:txtscore<="150";
151:txtscore<="151";
152:txtscore<="152";
153:txtscore<="153";
154:txtscore<="154";
155:txtscore<="155";
156:txtscore<="156";
157:txtscore<="157";
```

```
158:txtscore<="158";
159:txtscore<="159";
160:txtscore<="160";
161:txtscore<="161";
162:txtscore<="162";
163:txtscore<="163";
164:txtscore<="164";
165:txtscore<="165";
166:txtscore<="166";
167:txtscore<="167";
168:txtscore<="168";
169:txtscore<="169";
170:txtscore<="170";
171:txtscore<="171";
172:txtscore<="172";
173:txtscore<="173";
174:txtscore<="174";
175:txtscore<="175";
176:txtscore<="176";
177:txtscore<="177";
178:txtscore<="178";
179:txtscore<="179";
180:txtscore<="180";
181:txtscore<="181";
182:txtscore<="182";
183:txtscore<="183";
184:txtscore<="184";
185:txtscore<="185";
186:txtscore<="186";
187:txtscore<="187";
188:txtscore<="188";
189:txtscore<="189";
190:txtscore<="190";
191:txtscore<="191";
192:txtscore<="192";
193:txtscore<="193";
194:txtscore<="194";
195:txtscore<="195";
196:txtscore<="196";
197:txtscore<="197";
198:txtscore<="198";
199:txtscore<="199";
200:txtscore<="200";
201:txtscore<="201";
202:txtscore<="202";
203:txtscore<="203";
```

```
204:txtscore<="204";
205:txtscore<="205";
206:txtscore<="206";
207:txtscore<="207";
208:txtscore<="208";
209:txtscore<="209";
210:txtscore<="210";
211:txtscore<="211";
212:txtscore<="212";
213:txtscore<="213";
214:txtscore<="214";
215:txtscore<="215";
216:txtscore<="216";
217:txtscore<="217";
218:txtscore<="218";
219:txtscore<="219";
220:txtscore<="220";
221:txtscore<="221";
222:txtscore<="222";
223:txtscore<="223";
224:txtscore<="224";
225:txtscore<="225";
226:txtscore<="226";
227:txtscore<="227";
228:txtscore<="228";
229:txtscore<="229";
230:txtscore<="230";
231:txtscore<="231";
232:txtscore<="232";
233:txtscore<="233";
234:txtscore<="234";
235:txtscore<="235";
236:txtscore<="236";
237:txtscore<="237";
238:txtscore<="238";
239:txtscore<="239";
240:txtscore<="240";
241:txtscore<="241";
242:txtscore<="242";
243:txtscore<="243";
244:txtscore<="244";
245:txtscore<="245";
246:txtscore<="246";
247:txtscore<="247";
248:txtscore<="248";
249:txtscore<="249";
```

```
250:txtscore<="250";
251:txtscore<="251";
252:txtscore<="252";
253:txtscore<="253";
254:txtscore<="254";
255:txtscore<="255";
256:txtscore<="256";
257:txtscore<="257";
258:txtscore<="258";
259:txtscore<="259";
260:txtscore<="260";
261:txtscore<="261";
262:txtscore<="262";
263:txtscore<="263";
264:txtscore<="264";
265:txtscore<="265";
266:txtscore<="266";
267:txtscore<="267";
268:txtscore<="268";
269:txtscore<="269";
270:txtscore<="270";
271:txtscore<="271";
272:txtscore<="272";
273:txtscore<="273";
274:txtscore<="274";
275:txtscore<="275";
276:txtscore<="276";
277:txtscore<="277";
278:txtscore<="278";
279:txtscore<="279";
280:txtscore<="280";
281:txtscore<="281";
282:txtscore<="282";
283:txtscore<="283";
284:txtscore<="284";
285:txtscore<="285";
286:txtscore<="286";
287:txtscore<="287";
288:txtscore<="288";
289:txtscore<="289";
290:txtscore<="290";
291:txtscore<="291";
292:txtscore<="292";
293:txtscore<="293";
294:txtscore<="294";
295:txtscore<="295";
```

```
296:txtscore<="296";
297:txtscore<="297";
298:txtscore<="298";
299:txtscore<="299";
300:txtscore<="300";
301:txtscore<="301";
302:txtscore<="302";
303:txtscore<="303";
304:txtscore<="304";
305:txtscore<="305";
306:txtscore<="306";
307:txtscore<="307";
308:txtscore<="308";
309:txtscore<="309";
310:txtscore<="310";
311:txtscore<="311";
312:txtscore<="312";
313:txtscore<="313";
314:txtscore<="314";
315:txtscore<="315";
316:txtscore<="316";
317:txtscore<="317";
318:txtscore<="318";
319:txtscore<="319";
320:txtscore<="320";
321:txtscore<="321";
322:txtscore<="322";
323:txtscore<="323";
324:txtscore<="324";
325:txtscore<="325";
326:txtscore<="326";
327:txtscore<="327";
328:txtscore<="328";
329:txtscore<="329";
330:txtscore<="330";
331:txtscore<="331";
332:txtscore<="332";
333:txtscore<="333";
334:txtscore<="334";
335:txtscore<="335";
336:txtscore<="336";
337:txtscore<="337";
338:txtscore<="338";
339:txtscore<="339";
340:txtscore<="340";
341:txtscore<="341";
```

```
342:txtscore<="342";
343:txtscore<="343";
344:txtscore<="344";
345:txtscore<="345";
346:txtscore<="346";
347:txtscore<="347";
348:txtscore<="348";
349:txtscore<="349";
350:txtscore<="350";
351:txtscore<="351";
352:txtscore<="352";
353:txtscore<="353";
354:txtscore<="354";
355:txtscore<="355";
356:txtscore<="356";
357:txtscore<="357";
358:txtscore<="358";
359:txtscore<="359";
360:txtscore<="360";
361:txtscore<="361";
362:txtscore<="362";
363:txtscore<="363";
364:txtscore<="364";
365:txtscore<="365";
366:txtscore<="366";
367:txtscore<="367";
368:txtscore<="368";
369:txtscore<="369";
370:txtscore<="370";
371:txtscore<="371";
372:txtscore<="372";
373:txtscore<="373";
374:txtscore<="374";
375:txtscore<="375";
376:txtscore<="376";
377:txtscore<="377";
378:txtscore<="378";
379:txtscore<="379";
380:txtscore<="380";
381:txtscore<="381";
382:txtscore<="382";
383:txtscore<="383";
384:txtscore<="384";
385:txtscore<="385";
386:txtscore<="386";
387:txtscore<="387";
```

```
388:txtscore<="388";
389:txtscore<="389";
390:txtscore<="390";
391:txtscore<="391";
392:txtscore<="392";
393:txtscore<="393";
394:txtscore<="394";
395:txtscore<="395";
396:txtscore<="396";
397:txtscore<="397";
398:txtscore<="398";
399:txtscore<="399";
400:txtscore<="400";
401:txtscore<="401";
402:txtscore<="402";
403:txtscore<="403";
404:txtscore<="404";
405:txtscore<="405";
406:txtscore<="406";
407:txtscore<="407";
408:txtscore<="408";
409:txtscore<="409";
410:txtscore<="410";
411:txtscore<="411";
412:txtscore<="412";
413:txtscore<="413";
414:txtscore<="414";
415:txtscore<="415";
416:txtscore<="416";
417:txtscore<="417";
418:txtscore<="418";
419:txtscore<="419";
420:txtscore<="420";
421:txtscore<="421";
422:txtscore<="422";
423:txtscore<="423";
424:txtscore<="424";
425:txtscore<="425";
426:txtscore<="426";
427:txtscore<="427";
428:txtscore<="428";
429:txtscore<="429";
430:txtscore<="430";
431:txtscore<="431";
432:txtscore<="432";
433:txtscore<="433";
```

```
434:txtscore<="434";
435:txtscore<="435";
436:txtscore<="436";
437:txtscore<="437";
438:txtscore<="438";
439:txtscore<="439";
440:txtscore<="440";
441:txtscore<="441";
442:txtscore<="442";
443:txtscore<="443";
444:txtscore<="444";
445:txtscore<="445";
446:txtscore<="446";
447:txtscore<="447";
448:txtscore<="448";
449:txtscore<="449";
450:txtscore<="450";
451:txtscore<="451";
452:txtscore<="452";
453:txtscore<="453";
454:txtscore<="454";
455:txtscore<="455";
456:txtscore<="456";
457:txtscore<="457";
458:txtscore<="458";
459:txtscore<="459";
460:txtscore<="460";
461:txtscore<="461";
462:txtscore<="462";
463:txtscore<="463";
464:txtscore<="464";
465:txtscore<="465";
466:txtscore<="466";
467:txtscore<="467";
468:txtscore<="468";
469:txtscore<="469";
470:txtscore<="470";
471:txtscore<="471";
472:txtscore<="472";
473:txtscore<="473";
474:txtscore<="474";
475:txtscore<="475";
476:txtscore<="476";
477:txtscore<="477";
478:txtscore<="478";
479:txtscore<="479";
```

```
480:txtscore<="480";
481:txtscore<="481";
482:txtscore<="482";
483:txtscore<="483";
484:txtscore<="484";
485:txtscore<="485";
486:txtscore<="486";
487:txtscore<="487";
488:txtscore<="488";
489:txtscore<="489";
490:txtscore<="490";
491:txtscore<="491";
492:txtscore<="492";
493:txtscore<="493";
494:txtscore<="494";
495:txtscore<="495";
496:txtscore<="496";
497:txtscore<="497";
498:txtscore<="498";
499:txtscore<="499";
500:txtscore<="500";
501:txtscore<="501";
502:txtscore<="502";
503:txtscore<="503";
504:txtscore<="504";
505:txtscore<="505";
506:txtscore<="506";
507:txtscore<="507";
508:txtscore<="508";
509:txtscore<="509";
510:txtscore<="510";
511:txtscore<="511";
512:txtscore<="512";
513:txtscore<="513";
514:txtscore<="514";
515:txtscore<="515";
516:txtscore<="516";
517:txtscore<="517";
518:txtscore<="518";
519:txtscore<="519";
520:txtscore<="520";
521:txtscore<="521";
522:txtscore<="522";
523:txtscore<="523";
524:txtscore<="524";
525:txtscore<="525";
```

```
526:txtscore<="526";
527:txtscore<="527";
528:txtscore<="528";
529:txtscore<="529";
530:txtscore<="530";
531:txtscore<="531";
532:txtscore<="532";
533:txtscore<="533";
534:txtscore<="534";
535:txtscore<="535";
536:txtscore<="536";
537:txtscore<="537";
538:txtscore<="538";
539:txtscore<="539";
540:txtscore<="540";
541:txtscore<="541";
542:txtscore<="542";
543:txtscore<="543";
544:txtscore<="544";
545:txtscore<="545";
546:txtscore<="546";
547:txtscore<="547";
548:txtscore<="548";
549:txtscore<="549";
550:txtscore<="550";
551:txtscore<="551";
552:txtscore<="552";
553:txtscore<="553";
554:txtscore<="554";
555:txtscore<="555";
556:txtscore<="556";
557:txtscore<="557";
558:txtscore<="558";
559:txtscore<="559";
560:txtscore<="560";
561:txtscore<="561";
562:txtscore<="562";
563:txtscore<="563";
564:txtscore<="564";
565:txtscore<="565";
566:txtscore<="566";
567:txtscore<="567";
568:txtscore<="568";
569:txtscore<="569";
570:txtscore<="570";
571:txtscore<="571";
```

```
                        572:txtscore<="572";
                        573:txtscore<="573";
                        574:txtscore<="574";
                        575:txtscore<="575";
                        576:txtscore<="576";
                        577:txtscore<="577";
                        578:txtscore<="578";
                        579:txtscore<="579";
                        580:txtscore<="580";
                        581:txtscore<="581";
                        582:txtscore<="582";
                        583:txtscore<="583";
                        584:txtscore<="584";
                        585:txtscore<="585";
                        586:txtscore<="586";
                        587:txtscore<="587";
                        588:txtscore<="588";
                        589:txtscore<="589";
                        590:txtscore<="590";
                        591:txtscore<="591";
                        592:txtscore<="592";
                        593:txtscore<="593";
                        594:txtscore<="594";
                        595:txtscore<="595";
                        596:txtscore<="596";
                        597:txtscore<="597";
                        598:txtscore<="598";
                        599:txtscore<="599";
                        600:txtscore<="600";
                        601:txtscore<="601";


                endcase
        end




endmodule

//////////////////////////////
//
// KOOSH'S CODE
//
///////////////////////////////
```

```
//////////////////////////////////////////////////////////////////
//
//Analog Latch Reset - holds counter in reset state for 1/3 second
//                          this is time needed to reset the analog latch
//
//////////////////////////////////////////////////////////////////
module analog_latch_reset (clk, reset_lc, latch_reset, counter_reset);

      //reset_lc is enable signal for this module
      input reset_lc, clk;
      //counter reset resets counter module
      //latch_reset tied to output to analog circuitry
      output counter_reset, latch_reset;

      reg [23:0] count = 0;
      reg latch_reset = 1;
      reg counter_reset = 0;

      reg started;

      always @ (posedge clk) begin
            //wait for enable signal
            if (reset_lc) begin
                  count <= 0;
                  started <= 1;
                  end
     //count 1/3 second
            if (started) begin
                  count <= count + 1;
                  //latch reset opposite of counter reset
                  //latch resets on low input
                  latch_reset <= (count == 9000000) ? 1 : 0;
                  counter_reset <= (count == 9000000) ? 0 : 1;
            end
            if (count == 9000000) started <= 0;
      end

endmodule

//////////////////////////////////////////////////////////////////////
//
//steady_x_coord - gives enable when x has been ready and unchanging
//                    for 30 clock cycles. Compenstates for "ready"
//                    signals between calculations of d and y
//
//////////////////////////////////////////////////////////////////////
```

```
module steady_x_coord (clk, counter_ready, x_ready, x, y, x_steady, steady_x,
steady_y);
        input clk, counter_ready, x_ready;
        input signed [8:0] x, y;
        output x_steady;
        output signed [8:0] steady_x, steady_y;


        reg [5:0] steady_count = 6'd0;
        reg signed [8:0] old_x = 6'd0;
        reg signed [8:0] steady_x, steady_y;

        //x_steady after 30 cycles
        assign x_steady = (steady_count >= 30) ? 1 : 0;

        //waits for 30 consequtive consistant x values after "x_ready" and
"counter_ready"
        always @ (posedge clk) begin
          if (counter_ready & x_ready) begin
                old_x <= (steady_count == 6'd0) ? x : old_x;
                if (old_x == x)
                        steady_count <= (steady_count <=60) ? steady_count + 1 :
steady_count;
                else steady_count <= 0;
                if (steady_count >= 30) begin
                        steady_x <= x;
                        steady_y <= y;
                        end
                end
          //reset when old and current dont match
          else steady_count <= 0;

        end

endmodule

////////////////////////////////////////////////////////////////
//
//  dart register - holds values of 3 latest darts, controls
//                                  automatic reset and asserts data_ready when
//                                  3 darts have accumulated
//
///////////////////////////////////////////////////////////////
```

```
   module dart_register(clk, x_steady, x, y, x1, y1, x2, y2, x3, y3, reset_lc, data_ready,
data_taken, dart_count);
	input clk, x_steady, data_taken;
	input signed [8:0] x, y;
	output reset_lc, data_ready;
	output [1:0]dart_count;
	output signed [8:0] x1, y1, x2, y2, x3, y3;

	reg reset_lc;
	reg [1:0] dart_count = 2'b00;
	reg  signed [8:0] x1, y1, x2, y2, x3, y3;
	reg old_x_steady;
	reg data_ready, start_ready_count = 0;

	always @ (posedge clk) begin
		old_x_steady <= x_steady;
		reset_lc <= 0;
	   //reset values when darts taken by display logic
	   if (data_taken) begin
			dart_count <= 0;
			data_ready <= 0;
			reset_lc <= 1;
			x1 <= 9'b011100110;
			x2 <= 9'b011100110;
			x3 <= 9'b011100110;
			y1 <= 9'b011100110;
			y2 <= 9'b011100110;
			y3 <= 9'b011100110;
			end
	   //if this is new x_steady, save new dart information
	   else if ((x_steady != old_x_steady) && x_steady) begin
		case (dart_count)
			2'b00 : begin
					data_ready <= 0;
					//off screen darts moved to one location
					if ((x > 226) || (x < -226) || (y > 226) || (y < -226))
begin
						x1 <= 175;
						y1 <= 175;
						end
					else begin
						x1 <= x;
						y1 <= y;
						end
				   //increment dart count
					dart_count <= 2'b01;
```

```
                                        //reset latch and counter
                                        reset_lc <= 1;
                                        end
                        2'b01 : begin
                                        data_ready <= 0;
                                        if ((x > 226) || (x < -226) || (y > 226) || (y < -226))
begin
                                                x2 <= 175;
                                                y2 <= 175;
                                                end
                                        else begin
                                                x2 <= x;
                                                y2 <= y;
                                                end
                                        dart_count <= 2'b10;
                                        reset_lc <= 1;
                                        end
                        2'b10 : begin
                                        if ((x > 226) || (x < -226) || (y > 226) || (y < -226))
begin
                                                x3 <= 175;
                                                y3 <= 175;
                                                end
                                        else begin
                                                x3 <= x;
                                                y3 <= y;
                                                end
                                        dart_count <= 2'b11;
                                        reset_lc <= 1;
                                        //after dart three, data is ready to be taken
                                        data_ready <= 1;
                                        end
                        2'b11 : reset_lc <= 1;
                 endcase
                end
        end

endmodule

///////////////////////////////////////////////////////
//
// Cycles to Millimeters - This module converts cycles to
//                                      distances using a 3 bit shift
//
///////////////////////////////////////////////////////
  module cycles_to_mm (delta0, delta1, delta2, d0, d1, d2);
```

```
        //eight 2.7mhz cycles = 1 mm traveld by sound in air
        input [11:0] delta0, delta1, delta2;
        output [8:0] d0, d1, d2;

        wire [11:0] d0_temp, d1_temp, d2_temp;
        //shift by three approx divide by eight
        assign d0_temp = delta0>>3;
        assign d1_temp = delta1>>3;
        assign d2_temp = delta2>>3;
        //dont need first three bits anymore, always 0
        assign d0 = d0_temp[8:0];
        assign d1 = d1_temp[8:0];
        assign d2 = d2_temp[8:0];

endmodule

//////////////////////////////////////////////////////////////////////////
//
// Calculate X - calculates x position of dart given differeces in distances,
//                         distance to first mic and y position of dart
//
///////////////////////////////////////////////////////////////////////////
  module calc_x (d0, d1, d2, d, y, x, clock_27mhz, ready);
        input [8:0]d0,d1,d2;
        input signed [8:0]y;
        input [9:0]d;
        input clock_27mhz;
        output signed [8:0]x;
        output ready;

        wire [20:0] radical;
        wire [20:0] rad_term1;
        wire [19:0] rad_term2;
        wire signed [9:0]sqrt_term2;
        wire mul_ready;

        assign rad_term1 = (d+d0) * (d+d0);
        assign sqrt_term2 = y-200;
        //xilinx module used to do signed multiplication
        x_mul sqr_term2(clock_27mhz, sqrt_term2, sqrt_term2, rad_term2, mul_ready);
        //calculate value under radical
        assign radical = rad_term1 - rad_term2;
        wire signed [10:0] temp_x, neg_temp_x;
        //cordic module used to compute truncated square root
        x_sqrt1 gen_x(radical, clock_27mhz, mul_ready, temp_x, ready);
        //answer can be plus of minus, save minus value as well
```

```
	assign neg_temp_x = (0 - temp_x);
	//depeding on which half dart is on output correct +/- x
	assign x = (d1 <= d2) ? temp_x[8:0] : neg_temp_x[8:0];

endmodule

///////////////////////////////////////////////////////////////////
//
// Calculate Y - Calculates Y term based on d0, d1, d2 and d distance
//					to closest microphone
//
///////////////////////////////////////////////////////////////////
  module calc_y (d0, d1, d2, d, y, clock_27mhz);
	input [8:0] d0, d1, d2;
	input [9:0] d;
	output signed [8:0] y;
	input clock_27mhz;

	wire signed [21:0] term1;
	//first part of dividdend
	assign term1 = 80000 - (2*d*d0) - (d0*d0) + (2*d*d1) + (d1*d1);
	wire signed [20:0] sqrt_term1;
	wire signed [23:0] sqrt_term2;
	//values multiplied under the radical
	assign sqrt_term1 = 80000 - (d0*d0) + (2*d0*d1) - (d1*d1);
	assign sqrt_term2 = (4*d*d) - 80000 + (d0*d0) + (2*d0*d1) + (d1*d1) +
(4*d*d0) + (4*d*d1);

	wire [44:0] radical;
	wire mul_rdy;
	//xilinx multiplier used to do signed multiplication
	y_multiplier1 gen_radical(clock_27mhz, sqrt_term1, sqrt_term2, radical,
mul_rdy);

	wire [22:0] sqrt_out;
	wire sqrt_rdy;
	//cordic module used to compute truncated square root
	y_sqrt y_square_root(radical, clock_27mhz, mul_rdy, sqrt_out, sqrt_rdy);

	wire signed[23:0] dividend = term1 - sqrt_out;
	//dont use these remder
	wire [10:0] remd;
	wire [23:0] temp_y;
	//xilinx tool to do division
	y_div y_divider(.dividend(dividend), .divisor(11'b01100100000), .quot(temp_y),
.clk(clock_27mhz), .ce(sqrt_out));
```

```
        //only need last 9 bits
        assign y = temp_y[8:0];


endmodule

////////////////////////////////////////////////////////////////////////
//
//  Calculate D - compute d, distance to closest microphone, based on
//                              d0, d1, d2
//
////////////////////////////////////////////////////////////////////////

  module calc_d (d0, d1, d2, d, clock_27mhz);
  input [8:0] d0, d1, d2;
  output [9:0]d;
  input clock_27mhz;
  //basically implement quadratic equation
  wire signed [31:0]negb;
  wire [7:0] count_negb;
  wire [27:0] sqrt_out;
  wire [22:0] d_long;

  //this sub module does multiplication to determine first term
  math_negb math_mod1(d0, d1, d2, negb, count_negb, clock_27mhz);
  //this sub module does the square root to get the second part of the dividend
  math_sqrt math_mod2(d0, d1, d2, sqrt_out, clock_27mhz);
  //this module divides top term (negb - sqrt) by 2a
  math_divison math_mod3(d0, d1, d2, negb, sqrt_out, d_long, clock_27mhz);
  //d can only be 10 bits long in the real world
  assign d = d_long[9:0];  //know d will be less than 10 bits

endmodule

////////////////////////////////////////////////////////////////////////////
//
// Counter - Counts cycles of enable between mic1 mic2 and mic3 going high
//                      implemented as a fsm
//
////////////////////////////////////////////////////////////////////////////
  module counter (clock, mhz_enable, reset, mic1, mic2, mic3, d1, d2, d3, ready, state);
        input clock, mhz_enable, mic1, mic2, mic3, reset;
        output[11:0] d1, d2, d3;
        reg [11:0] d1 = 0, d2 = 0, d3 = 0;
        output ready;
        output [2:0]state;
```

```
	reg ready = 0;
	reg [2:0]state = 0;
	//each state represents a combination of the 3 counters counting
	parameter RESET_STATE = 3'b000;
	parameter MIC12COUNT  = 3'b001;
	parameter MIC13COUNT  = 3'b010;
	parameter MIC23COUNT  = 3'b011;
	parameter MIC1COUNT   = 3'b100;
	parameter MIC2COUNT   = 3'b101;
	parameter MIC3COUNT   = 3'b110;
	parameter READY_STATE = 3'b111;

	always @ (posedge clock) begin
		if (mhz_enable) begin
			case (state)
				//in reset keep all counts at 0 wait for a mic to go high
				RESET_STATE : begin
							d1 <= 0;
							d2 <= 0;
							d3 <= 0;
							ready <= 0;
							//depending on which mic(s) go high
go to appropriate count state
							state <= reset ? RESET_STATE
:(mic1 & mic2 & mic3) ? READY_STATE :
									(mic1 & mic3) ?
MIC2COUNT : (mic1 & mic2) ? MIC3COUNT :
									(mic2 & mic3) ?
MIC1COUNT : mic1 ? MIC23COUNT : mic2 ? MIC13COUNT :
									 mic3 ?
MIC12COUNT : RESET_STATE;
						end
				//mics 1 and 2 counting
				MIC12COUNT   : begin
							 d1 <= d1 + 1;
							 d2 <= d2 + 1;
							 //go ready if all mics go high
							 state <= reset ? RESET_STATE :
								(mic1 & mic2) ?
READY_STATE : mic1 ? MIC2COUNT : mic2 ? MIC1COUNT : MIC12COUNT;
							end
				//mics 1 and 3 counting
				MIC13COUNT   : begin
							 d1 <= d1 + 1;
							 d3 <= d3 + 1;
							 //go ready if all mics go high
```

```
                                        state <= reset ? RESET_STATE :
                                                (mic1 & mic3) ?
READY_STATE: mic1 ? MIC3COUNT : mic3 ? MIC1COUNT : MIC13COUNT;
                                        end
                        //mics 2 and 3 counting
                        MIC23COUNT   : begin
                                        d2 <= d2 + 1;
                                        d3 <= d3 + 1;
                                        //go ready if all mics go high
                                        state <= reset ? RESET_STATE :
                                                (mic2 & mic3) ?
READY_STATE : mic2 ? MIC3COUNT : mic3 ? MIC2COUNT : MIC23COUNT;
                                        end
                        //only mic 1 counting
                        MIC1COUNT    : begin
                                        d1 <= d1 + 1;
                                        //go ready if all mic 1 goes high
                                        state <= reset ? RESET_STATE :
mic1 ? READY_STATE : MIC1COUNT;
                                        end
                        //only mic2 counting
                        MIC2COUNT    : begin
                                        d2 <= d2 + 1;
                                        //go ready if all mic 2 goes high
                                        state <= reset ? RESET_STATE :
mic2 ? READY_STATE : MIC2COUNT;
                                        end
                        //only mic3 counting
                        MIC3COUNT    : begin
                                        d3 <= d3 + 1;
                                        //go ready if all mic 3 goes high
                                        state <= reset ? RESET_STATE :
mic3 ? READY_STATE : MIC3COUNT;
                                        end
                        //output ready when all mics done counting
                        READY_STATE  : begin
                                        ready <= 1;
                                        state <= reset ? RESET_STATE :
READY_STATE;
                                        end
                        endcase


        end
        //go to reset state even in reset signal comes in between enable signals
        if (reset) state <= RESET_STATE;
```

```
           end

   endmodule

////////////////////////////////////////////////////////////////////////
//
// Math Negative B - Compute the negative b term of the calculate d equation
//
/////////////////////////////////////////////////////////////////////////

   module math_negb(d0, d1, d2, negb, count, clk);
           input clk;
           input [8:0] d0, d1 ,d2;
           output signed [31:0] negb;
           output [7:0] count;
           reg [7:0]count;
           reg signed [31:0]old_negb = 0;

           wire signed [31:0] negb;
           //simply * to compute term
           assign negb =  (80000 * d1) - (2*d0*d0*d0) + (d0*d0*d1) + (d0*d1*d1) -
(d1*d1*d1) + (80000*d2) + (d0*d0*d2) + (d0*d2*d2) - (d2*d2*d2);
           //count how many clock cycles befor valid
           always @(posedge clk) begin
                     count <= (old_negb != negb) ? count + 1 : count;
                     old_negb <= negb;
                     //count <= 1;
           end

endmodule

////////////////////////////////////////////////////////////////////////
//
// Math Negative B - Do division part of calculate d equation
//
/////////////////////////////////////////////////////////////////////////

   module math_divison (d0, d1, d2, negb, sqrt_out, d, clk);
     input clk;
           input [8:0] d0, d1, d2;
           input [31:0] negb;
           input [27:0] sqrt_out;
           output [31:0]d;

           wire signed [31:0]top;
```

```
	//dividend
	assign top = negb - sqrt_out;

	wire signed [22:0]divisor;
	assign divisor = (-80000 + (2*d0*d0) + (d1*d1) + (d2*d2) - (2*d0*d1) -
(2*d0*d2))<<<1;
	wire rfd;
	wire [22:0]remd;
	//xilinx tool to do division
	divide div(top, divisor, d, remd, clk, rfd);

  endmodule

///////////////////////////////////////////////////////////////////////
//
// Math Square Root - Compute the square root of term in the calculate d equation
//
///////////////////////////////////////////////////////////////////////

  module math_sqrt (d0, d1, d2, sqrt_out, clk);
	input clk;
	input [8:0] d0, d1, d2;
	output [27:0]sqrt_out;

	wire [17:0] d0_sqr, d1_sqr, d2_sqr;
	wire [19:0] d0d1, d0d2, d1d2;

	//terms in equation
	assign d0_sqr = d0*d0;
	assign d1_sqr = d1*d1;
	assign d2_sqr = d2*d2;
	assign d0d1 = 2*d0*d1;
	assign d0d2 = 2*d0*d2;
	assign d1d2 = 2*d1*d2;

	wire signed [16:0] term1, term2, term3;
	//do shifts to make math easier/use less bits
	assign term1 = (20000 - d0_sqr[17:2] + d0d1[19:4]*4 - d1_sqr[17:2]);
	assign term2 = (20000 - d0_sqr[17:2] + d0d2[19:4]*4 - d2_sqr[17:2]);
	assign term3 = (40000 - d1_sqr[17:2] + d1d2[19:4]*4 - d2_sqr[17:2]);

	wire [50:0]sqrt_in_long;
	wire [33:0] term1_term2;
	wire mult1ready, mult2ready;
	//multipl term1 and term 2 and term 3 using cascaded multipliers
	multiplier1 term1term2(clk, term1, term2, term1_term2, mult1ready);
	multiplier2 final_mul(clk, term1_term2, term3, sqrt_in_long, mult1ready,
mult2ready);

	wire [47:0]sqrt_in_trunc;
	assign sqrt_in_trunc = sqrt_in_long[47:0];  //no way to get more than 48bits
	                                                                                  //only have it
at 51 because of twos complement
	wire [24:0]sqrt_out_short;
	wire ready;
	//use cordic to take square root of long equation
   d_sqrt my_sqrt (sqrt_in_trunc, clk, mult2ready, sqrt_out_short, ready);
	assign sqrt_out = sqrt_out_short<<3;
	endmodule


////////////////////////////////////////////////////////////////////////
//
// Divider - Provide 2.7 mhz enable signal
//
////////////////////////////////////////////////////////////////////////
  module divider (clock, mhz_enable);
	input clock;
	output mhz_enable;
	reg[4:0] count = 0;
	//count from 0 - 9 = 10 cycles
	assign mhz_enable = (count == 9) ? 1 : 0;

	always @ (posedge clock) begin
		count <= (count == 9) ? 0 : count + 1;
	end
  endmodule
```