Immersive 3D World

MIT 6.111 Fall 2015

Project Report

Christine Konicki and Mikhail Rudoy

Professor: Gim Hom

December 9, 2015

Table of Contents

Introduction

Virtual reality has become a new and exciting realm for research and product development. This past July, OnePlus became the first company to launch a product using VR, and others are slowly incorporating virtual reality into what they build. Our final project uses the 6.111 labkit (and FPGA) to create an audio-visual virtual reality which responds to the turning motion(s) of the user's head. In our project, the user is able to walk through a virtual world—as displayed on a screen via VGA—using buttons on the labkit. The turning of the user's head triggers a "looking around" effect as if the user were actually in the virtual world looking around; for example, looking to the right causes the contents of the screen to shift left.

The virtual 3D world is constructed using a triangle model where every triangle is allocated hardware within the FPGA. The graphics system allows the player to translate and rotate her in-world perspective. This motion in the virtual world is at the hands of the player through the labkit buttons. In addition, the player is expected to wear a headphone set with an attached 3-axis gyroscope. The gyroscope picks up rotation values in multiple directions for the player's head, and these readings are used to adjust the player's perspective in the 3D world so as to accomplish the "looking around" effect described above. The headset to which the gyroscope is attached is also used to play the sound of footsteps as the user walks around in the virtual world.

Project Overview

Our project consists of six parts: a gyroscope interface module, the user input interface, an audio module, a view angle computation module, a finite state machine (FSM) for tracking the user's position in the virtual world, and a graphics engine for VGA. The overall design, consisting of these parts, is shown in the diagram in Figure 1.

The gyroscope interface is shown in the top left of the block diagram. It uses an Arduino to configure the gyroscope upon system startup, to read from the gyroscope when prompted, and to send bit streams of the readings in series form to the FPGA. The Arduino also processes the gyroscope readings, adjusting for bias and retransmitting the readings a byte at a time. In the

FPGA, an interface module requests gyroscope readings at a rate of 60Hz (the video frame rate) and converts the serial responses into simultaneously available parallel signals. The request is timed to occur immediately before the start of every new video frame according to the video signal vsync.

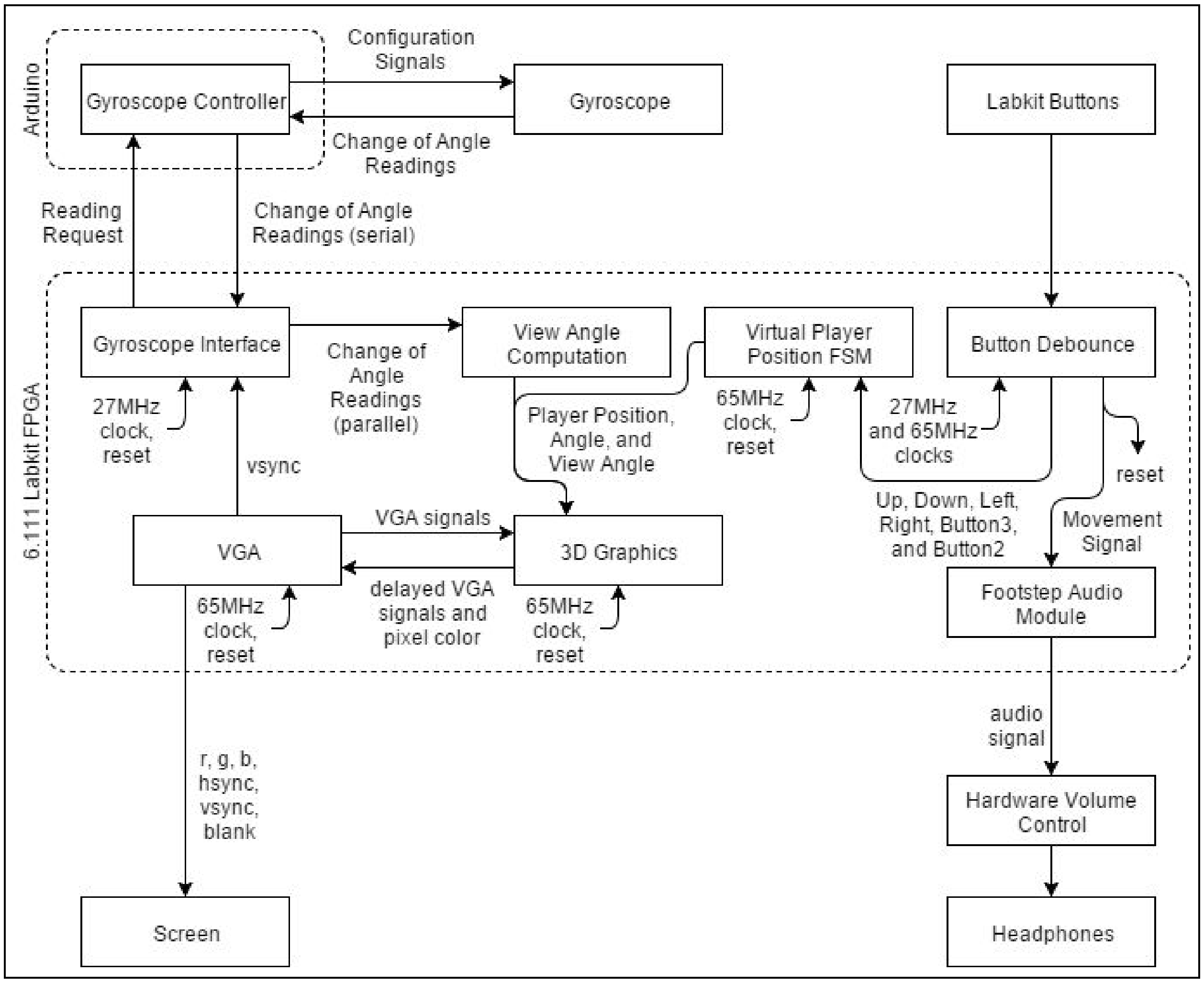

Figure 1*: A block diagram showing the overall design of our project.*

The view angle computation module takes as input the readings collected in the gyroscope interface and turns them into an estimate of the user's viewing angle. This angle is estimated relative to a fixed reference point in which the user is directly facing the center of the screen. Since the values outputted by the gyroscope are changes in angle (since the last reading), this module mainly acts as an accumulator of the gyroscope readings, integrating the gyroscope's changes in angle to compute a viewing angle in each of the three directions. This direct approach of simply accumulating readings has several downsides including gyroscope drift and the need for a configuration step in which the user indicates the gyroscope position of the reference point (when facing the center of the screen). Our solution to these problems is to accumulate the values with an exponentially weighted moving average. Under the assumption that the user generally faces the screen, this solution both fixes the gyroscope drift and removes the need for configuration. In fact, this solution even allows the user to change their seat or adjust the angle at which they wear their headphones without needing an explicit reconfiguration step.

The user input interface (UI), shown in the top right of Figure 1, debounces the signals from the labkit buttons. These debounced signals are sent to the other modules to indicate which of the buttons were pressed by the user. In addition, the signal from the "enter" button is combined with the labkit reset signal to generate an overall reset signal used by many of the other modules. One detail obscured in the block diagram is that two different reset signals are generated: one for the modules synchronized to a 65MHz clock and one for the modules synchronized to a 27MHz clock.

The virtual player position FSM tracks the user's position and orientation in the virtual world. The user's virtual position is restricted to the (horizontal) x-z plane. Thus the user's position consists of three values: the user's x-coordinate, the user's z-coordinate, and the user's angle of rotation in the horizontal plane. The module adjusts the angle of rotation up or down according to the state of two of the labkit buttons (as indicated by the UI module). In addition, the four direction buttons (up, down, right, and left) control motion in the horizontal plane. The up and down buttons control movement in the direction the user is facing while the right and left control movement in the orthogonal direction. This is made possible by using a trig module which computes the sine and cosine of the user's angle.

The audio module, depicted in the bottom right of Figure 1, uses the signals from the UI module to determine whether the user is pressing any of the buttons controlling player movement. Whenever the user is moving in the virtual world, this module triggers a low-frequency square wave which is sent outside the FPGA. This signal is scaled down in voltage in hardware (corresponding to a volume control) and played out through the user's headphones. The result is a periodic ticking noise which simulates the sound of footsteps.

The graphics engine, shown in the bottom left of Figure 1, consists of two parts. First is the VGA module which generates the necessary signals at the correct timing for 1024-by-768 resolution VGA. This module was given to us in 6.111 for the pong-game lab, and we used it exactly as given. The second part is mainly used to generate the color values for the pixels. The color computation is quite complex and takes longer than one clock cycle; as a result we had to pipeline the computation. In order for the timing to line up, the module also outputs a delayed version of the VGA timing signals whose delay matches the pipeline delay. These signals are sent to the external screen. The actual color computation consists of several steps. First the vertices of the triangles are rotated and translated into a new coordinate system according to the player position and angle (from the FSM) and also according to the user view angles (from the view angle module). In the resulting coordinate system, the endpoints have three coordinates, of which two correspond directly to coordinates on the screen, and the last corresponds with the point's depth into the screen. After this is done, a signal is generated for each triangle indicating whether the current pixel on screen is "inside" the triangle, and if it is, the depth of the point into the screen is also computed. Next, these signals are recombined to get a single color and a single depth: the color of the closest triangle and the depth into the screen at which that triangle can be found. Finally, the color is adjusted according to the depth resulting in an effect where objects fade as they move into the distance. This final color is used as the output of the module.

The Verilog code for all of these parts is attached in Appendix A. The Arduino code for the gyroscope interface is attached in Appendix B.

Changes from the Proposal

Throughout the project, there were several times when we chose to deviate from the originally proposed design. We made these design changes in our project to deal with practical problems as they came up. Most of the design, however, remained the same.

The first difference is in the gyroscope interface. We originally planned to implement the system entirely in Verilog, but we eventually realized that since the gyroscope chip was originally designed for Arduino, it would be much easier to configure and later read from using an Arduino.

The second difference is in the UI module. We originally planned to allow the user to control movement using a game controller, but we underestimated the complexity of the circuitry involved. We could not find sufficient information about the controller which we purchased; as a result, we were not able to track the button presses and joystick motions. Instead we switched to a fallback design and used the FPGA buttons instead.

The last difference was in the view angle computation module. This difference was due to a misunderstanding of the function of a gyroscope: we originally believed that the gyroscope would output angle readings indicating its angle in the coordinate system of the gyroscope's startup reference frame. In reality, the gyroscope outputs changes in coordinates in the coordinate system of its instantaneous local reference frame. The view angle computation module, which was originally named the gyroscope coordinate transform module, was supposed to take as input several reference readings (i.e. with the user facing the left and right edges of the screen) as well as the current readings. It would then use a change of coordinates to output a transformed version of the current readings indicating the angular position of the user's head.

The module was supposed to use a configuration step to deal with two issues: the unknown orientation of the gyroscope and the constant offset in all of the readings due to an overall change in angle from the gyroscope's initial reference frame. Since the gyroscope readings are changes in angle rather than the angles themselves, the second issue was immediately resolved. Thus the coordinate transformation was only necessary to deal with the unknown orientation of the gyroscope. This coordinate transformation turned out to be quite complicated mathematically, and we were able to avoid the difficult computation by making a reasonable assumption about the use cases that would come up.

Modules in Detail

Below, we will describe each of the parts of the project, one at a time, in more detail.

Gyroscope Interface (Christine)

Using Verilog (Original Approach)

Initially, it was our intention to implement the entire gyroscope interface in Verilog. An additional Verilog module would be created to write the relevant control bits serially to the L3GD20 gyroscope (Figure 2) in order to configure it properly. To write information to the chip, the gyroscope would be configured for SPI communication by grounding the CS pin (Figures 2 and 3) and activating the serial clock pin (SCL) with a 9MHz clock. The serial data input pin (SDI/SDA) would then receive the following 16-bit signal: a 0 indicating that we are writing to the gyroscope, a 0 indicating we only wish to write to one register (a 1 would indicate that the register address should be incremented so we could write to the next one as well), the 6-bit address of the register we wish to write to, and 8 data bits to be written to the register. Once all 16 bits have been sent, CS would be set high again, and the serial clock would be halted.

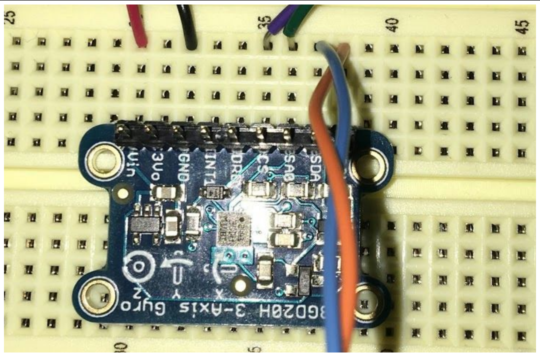

Figure 2: *The L3GD20 gyroscope chip before being attached to the headphones and wired to the Arduino.*

The first step in configuring the gyroscope for general operation would be to set the gyroscope to "normal mode" (i.e. power it on) and enable the x, y, and z channels for recording and sending readings. This is done by writing to the appropriate control register in the gyroscope (CTRL_REG1). The other four CTRL_REG registers remained untouched, as their default values were already correct for general operation of the gyroscope (see Appendix B).

To read changes in the x, y, and z coordinates from the gyroscope, we must again set up the chip for serial communication by grounding CS and activating SCL with the 9MHz clock. SDI would then receive an 8-bit signal: a 1 indicating that we are reading from the gyroscope, a

0 to read from only one register, and the 6-bit address of the register we wish to read from. The byte stored at the register would then be outputted serially through the serial data output pin (SDO/SA0). Each of the three coordinate readings measured by the gyroscope is 16 bits long with each byte stored in a different register (i.e. two registers for one reading), which totals to six registers for all three readings.

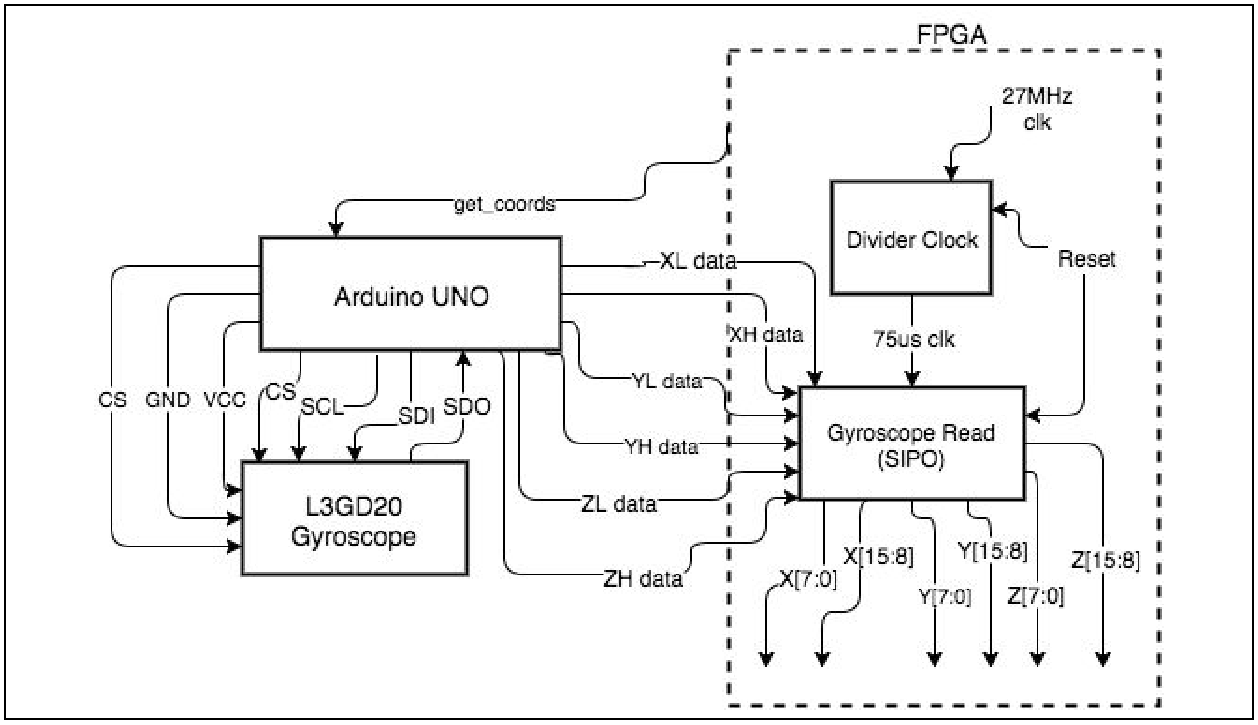

***Figure 3**: The gyroscope interface module*

Implementing the reading and writing processes in Verilog was ultimately unsuccessful because there was no fool-proof way to check that our readings were correct and that our write operations were completed correctly. There were also several strange occurrences such as output appearing at SDO during a write operation (which is forbidden according to the L3GD20's datasheet) and a lack of SDO output at the appropriate stage during a read operation. Further investigation suggested that interfacing the gyroscope with an Arduino was much more practical and likely to work, particularly since the chip's manufacturer, Adafruit, supplied an Arduino library that configured the chip's control registers automatically and provided read and write helper methods (Appendix B).

Using Arduino (Actual Approach)

The first thing we did to interface the Arduino with the gyroscope was wire the power, ground, CS, SCL, SDI and SDO pins of the gyroscope to the power, ground, and various digital pins of the Arduino. Next, we downloaded the Adafruit open-source L3GD20 configuration library to our Arduino module, which reduced the configuration process to a simple library import. We then initialized additional digital pins on the Arduino to be serial outputs to the FPGA, along with an additional digital pin as an input from the FPGA called "get_coords" (see Figure 3). This periodic trigger indicated to the gyroscope that the current readings should be sent serially to the FPGA, one byte per pin for a total of six digital output pins (e.g. XL outputs the lower byte of the x-coordinate, XH outputs the upper byte of the x-coordinate, YL outputs the lower byte of the y-coordinate, etc.).

Inside the FPGA were a series of six Verilog read modules, one for each of the incoming coordinate bytes, which converted the serial inputs from the Arduino into parallel bytes of data (see x[15:0], y[15:0], z[15:0] in Figure 3) to be utilized by the other modules. The conversion was similar to the serial-to-parallel conversion performed in the infrared lab. In that lab, the signal from the remote control was oversampled at a rate of 75 microseconds to compute the number of ones over a given period, determine if the current bit being sent was a 0 or a 1, and ultimately convert the learned bits into a 12-bit signal. The Arduino produced a similar signal for each coordinate byte, and the FPGA read module learned the received bits in the same fashion. To produce a clock with a 75 microsecond period from a 27 MHz clock (which has a smaller period), we also introduced a clock divider in Verilog.

View Angle Computation (Mikhail)

Module overview

The role of the view angle computation module is to convert the gyroscope readings from the gyroscope interface into view angles. The two angles of interest are shown in Figure 4. In each of the images we see the screen together with a reference line from the location of the user to the center of the screen. The red line (shown on the left of the figure) is the viewing vector: the line along which the user is looking. This vector can be decomposed along the screen into vertical and horizontal components. The angles associated with these components, shown in green and blue in the center and on the right, are the view angles we are interested in.

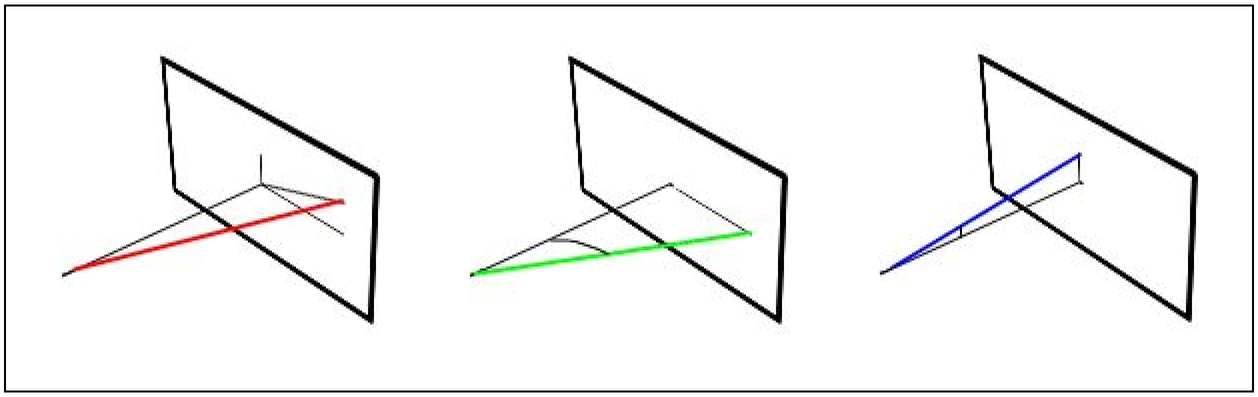

***Figure 4**: The view angles associated with a given viewing vector (red).*

The gyroscope, however, does not output these angles directly. Rather, the gyroscope outputs the changes in angle from the last measurement along three directions. The three angles are called yaw, pitch and roll. Figure 5 demonstrates the axes of rotation for each of these three measures. For the user's head, pitch corresponds to the motion of nodding, yaw to the motion of turning the user's head, and roll to the motion of tilting the user's head side to side. Looking at the diagram, it is clear that if the user faces the screen, the two angles we care about are yaw and pitch.

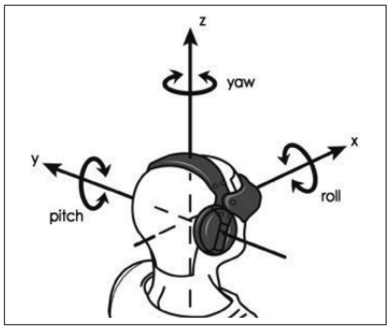

***Figure 5**: Roll, pitch, and yaw for a user's head.*

Unfortunately, the values we need (the view angles) are equal to the yaw and pitch of the user's head, not the yaw and pitch of the gyroscope. Thankfully, we can simplify the problem by making a few useful simplifying assumptions. We assume that the user will generally be (at least approximately) facing the screen (more specifically the center of the screen), that the user's head will be vertically aligned with the vertical direction of the screen, and that she will be wearing the headphones in a reasonable manner. In this case, a "reasonable manner" means that the headphones are worn in the correct direction (the right side on the right and the left side on the left) with the strap over and/or behind the head at some angle.

This final assumption helps a lot in simplifying the required math. Consider Figure 6, which shows the local coordinates for the user's head and for the gyroscope, which is located at the top of the headphone's strap. As the figure shows, the green x axes (for the head and the gyroscope) are parallel, and so the pitch is exactly the same for the user's head as it is for the gyroscope. This allows us to easily compute the pitch of the user's head: simply use the pitch of the gyroscope.

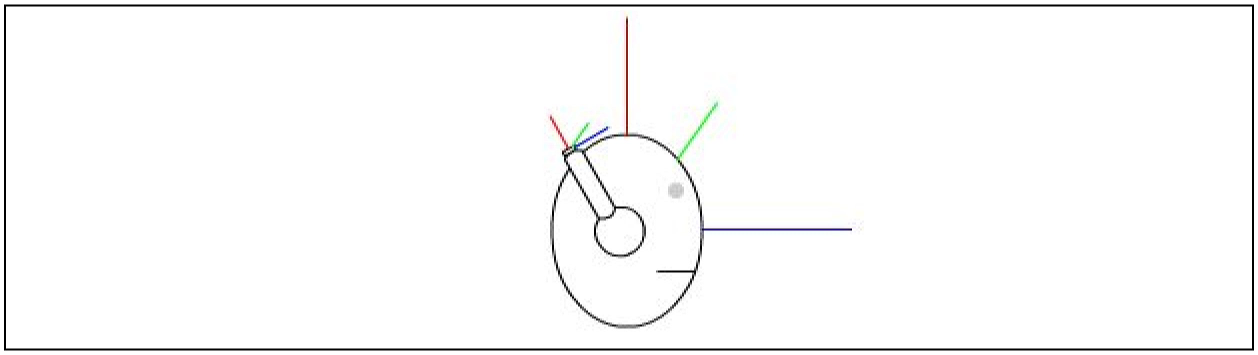

Figure 6*: A side view of the user's head. Local coordinate axes are shown for both an example user's head and the gyroscope at the top of the headphone's strap.*

On the other hand, the blue z and red y axes do not line up perfectly. We wish to compute (or at least approximate) the yaw of the user's head, which corresponds to the measure of rotation around the red y axis. At one extreme, the user might be wearing the headphones all the way up, in which case the gyroscope axes are exactly parallel to the user's head's axes; in this case, the yaw of the user's head is exactly the yaw of the gyroscope. At the other extreme, the user might be wearing the headphones all the way back, in which case the gyroscope's blue axis becomes parallel with the user's head's red axis; in this case, the yaw of the user's head is exactly the roll of the headphones. In the general case, the yaw of the user's head can be computed as some combination of the yaw and pitch of the gyroscope. Since we hardly needed this value to be exact, we simply approximated the yaw of the user's head as the yaw of the gyroscope added to the roll of the gyroscope.

This is already a big change from the original design. In the original design, we were planning to compute the change of coordinates from the gyroscope's reference frame to the user's head's reference frame exactly. This original design is much more complex than the design outlined above, but can allow for more general use. We decided, however, that the tradeoff was worth it: the decrease in complexity due to the change is very significant, while the decrease in usability is minor. Almost every use-case that a real user would want (every reasonable way of wearing headphones while remaining upright and facing the screen) is handled correctly, and only other less common uses (i.e. head tilted to the side) fail.

Computing the Angles

Above, we developed a method of approximately computing the view angles: compute the gyroscope's yaw, pitch, and roll, and output pitch and yaw plus roll. This still requires some work though since the gyroscope outputs the changes in these values from the last request rather than outputting the values directly. This is another difference from the originally proposed design. In the original proposal, we did not realize that this is how gyroscopes work, and thought that the gyroscope outputs yaw, pitch, and roll directly.

Since the values we read in are changes in the values we actually want to compute, a naive idea would be to simply accumulate the sum of the readings so far, leading to estimates of the yaw, pitch, and roll. There are many problems with this approach.

First of all, the changes in yaw, pitch, and roll reported by the gyroscope are in the instantaneous local reference frame of the gyroscope, which itself is changing as the gyroscope moves. Thus simply accumulating each value independently ignores the dependence between the values. For example, 90 degrees of yaw, followed by 90 degrees of roll, followed by 270 degrees of yaw is exactly the same rotation as 90 degrees of pitch; a simple accumulation of the values, however, fails to capture this equivalence. The only way to accurately capture the orientation implied by the readings is to keep an internal representation of the gyroscope's orientation and update it according to the roll, pitch, and yaw in the current local frame.

Even if we were to do this (rather involved) calculation, however, we would come upon a second problem: gyroscope drift. Gyroscope drift is the phenomenon where the value of the angles observed when the gyroscope repeatedly returns to the same position (i.e. with the user directly facing the center of the screen) drifts over time away from the correct value. This phenomenon occurs simply because of accumulation of error, and is commonly observed in many if not all applications involving gyroscopic sensors.

Finally, there is a third problem inherent with this approach. The value of zero for pitch, yaw, and roll is assigned to the position of the gyroscope upon startup of the accumulation. Thus to achieve the desired behavior, we would have to have a configuration step, in which the user is asked to look at the center of the screen before the accumulation begins. Even if we do that, however, the system would have to be reconfigured whenever anything changes. If, for example, the user were to shift the headphones from one reasonable position to another, there would be a corresponding change in pitch, which would be sensed by the gyroscope and permanently added to the accumulated pitch value; as a result, the user's virtual avatar would be permanently stuck either looking up or looking down. As another example, consider the situation in which the user switches seats. This change is likely accompanied by a change in angle of the user's view to the screen; most likely this change is measured as a change of yaw. The user's virtual avatar would then be stuck permanently looking right or left.

Our solution is to not accumulate the values directly but instead to use an exponentially weighted moving average of the readings. As it turns out, given our assumptions, this solution fixes all of the above problems. In an exponentially weighted moving average, the previous readings from the gyroscope are all averaged with the weights for the average exponential in the amount of time since the reading. One way to accomplish this is to maintain an accumulated value; at each new reading, compute the new value for the accumulator by multiplying the accumulated value by a constant factor between zero and one and adding the new reading.

The correct function of this algorithm relies entirely on our assumptions: that the user generally looks at the screen and more specifically at the center of the screen. Under this assumption, the user's head is always facing roughly in the same direction, and as a result, the local reference frame for the gyroscope remains roughly constant. Since the gyroscope's local coordinate system does not move much, we do not have to worry about the interdependence between the yaw, pitch, and roll, and can simply accumulate them independently. Whatever effect we actually observe due to this dependence can simply be treated as another source of gyroscope drift, which is also fixed by using an exponentially weighted moving average.

The assumption that the user generally (or on average) faces the center of the screen gives us a natural reference point for the gyroscope heading. Whenever the user looks directly at the center of the screen, we can associate that with yaw, pitch, and roll values that are all zero. When using the naive accumulation method, there was no reason for this to be the case; any other values would work just as well, and so over time the values could be free to drift. In our implementation, however, the weight assigned to any given reading decreases exponentially over time, and so the accumulated values will always go to zero unless presented with continuous non-zero readings which do not cancel out. Since the user generally faces only a small range of possible directions (the user faces the screen), readings will generally cancel out (looking at one edge of the screen and then the opposite will result in first one reading and then its negation); thus “continuous non-zero readings which do not cancel out” will not occur, and the accumulated values will generally tend to zero. As a result, gyroscope drift cannot occur since the values will always return to (approximately) zero.

The interpretation for the algorithm is this: whenever the user looks in a single general direction for a while, the accumulated values all tend towards zero. At that point, the general direction in which the user was looking is now the new zero point and is interpreted as the direction in which the center of the screen lies. After that, and short term movements (i.e. glancing to the right and then looking back left) will yield the expected results, since in the short term all of the weights are approximately one and so the accumulator behaves similarly to the naive accumulator. Long term changes, however, result in adjustments of the zero-point, as warranted by the assumption that the user looks at the center of the screen. Such long term changes can be the results of a user changing seats or adjusting the position of the headphones. Clearly, no special configuration step is necessary to set the reference point (simply looking at the screen for a while sets that reference) and no reconfiguration step is necessary if anything changes (again, simply looking at the screen is enough to automatically reconfigure).

Implementation

For our actual implementation, the choice of decay parameter for the exponentially weighted average was simple: we wanted the half-life of the decay to be about ten seconds (so that looking at the screen for about 10 seconds would mostly reset the accumulators to zero). Since the frame rate is 60Hz and the gyroscope is being read once per frame, the accumulated value should halve in 600 frames. In other words, the decay parameter d satisfies $d^{600} = 0.5$, or in other words d is approximately 0.998845422. We converted this value to binary and used that as a starting point; we later made small adjustments to the value to improve the functionality of the system.

As a result of the simplifying assumptions, actually implementing the module was relatively fast and painless. A good takeaway is that one should make simplifying assumptions in the design process whenever they are warranted. Avoiding that simplification for an improvement in functionality for a special use-case that will never occur is a waste of time and effort.

One final issue we encountered was that the gyroscope readings were noisy. To avoid this issue, we used an average of the last few readings in the accumulation step rather than just using the readings themselves. In effect, we smoothed the readings with an averaging filter before using them in the accumulators.

User Interface (Christine)

The user interface has two primary components for exploring the virtual world: the headphones and the FPGA buttons. The headphones enable tracking of the turning of the user's head, and the buttons enable translation and rotation in the world.

The headphones are used as a convenient bridge between the gyroscope and the user (Figure 7). The gyroscope is wired to a breadboard, and the breadboard is attached to the headband of the headphones with duct tape. The wires connecting gyroscope pins to the Arduino pins come out of the breadboard and are taped to the headband, ear muff, and cords to keep them out of the way of the user. Because of the headphones, the gyroscope can directly register the turning of the user's head without the user having to do anything.

The FPGA buttons were supposed to be replaced with the two joysticks (one for translation, the other for rotation) of a video game controller for added complexity and intrigue. However, we were unable to find sufficient information about the inner circuitry of the controller to determine which signals were connected to the joysticks and how to interpret them. We replaced the two joysticks with six FPGA buttons. The "up", "down," "left," and "right" buttons were used for translation in the x-z plane of the virtual world, and buttons two and three were used for rotation about the y-axis of the virtual world. All of the buttons were debounced using the Verilog debounce module from the pong-game lab.

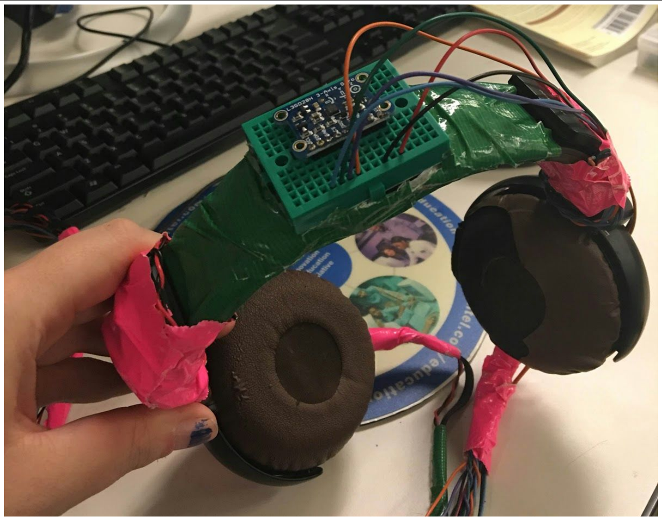

Figure 7: *The L3GD20 gyroscope wired to a green breadboard on the headphones. The breadboard is sturdily attached the headphones, and the wires connecting the breadboard to the Arduino are taped to the headphone band and cords to prevent them from impeding the user's interaction with the system.*

Virtual Player Position FSM (Mikhail)

For this module, we used inputs from the UI module to control the x, z, and angle positions of the user in the virtual world. Note that the player only moves in the horizontal x-z plane; in other words the player can move move left and right, move into and out of the screen, and can rotate within this plane, but cannot move up and down. The three coordinate of the user's position are shown in Figure 8. Note that this module has a simplified role (compared to the role in the original proposal) given that the view angle computation no longer requires a configuration step.

The player's button presses could lead to translation or rotation in this plane. In particular, buttons two and three control rotation while the "up", "down", "left", and "right" buttons control translation. Buttons two and three directly increment and decrement the player's angle. The translation, on the other hand, is a little more complicated. Pressing up leads the user to move in the direction she is facing. In Figure 8, for example, the user would move in the direction of the blue arrow. The other buttons cause the user to move left, right, or down according to the way the user is facing (i.e. the down button causes the user to move opposite of the direction of motion caused when the up button is pressed).

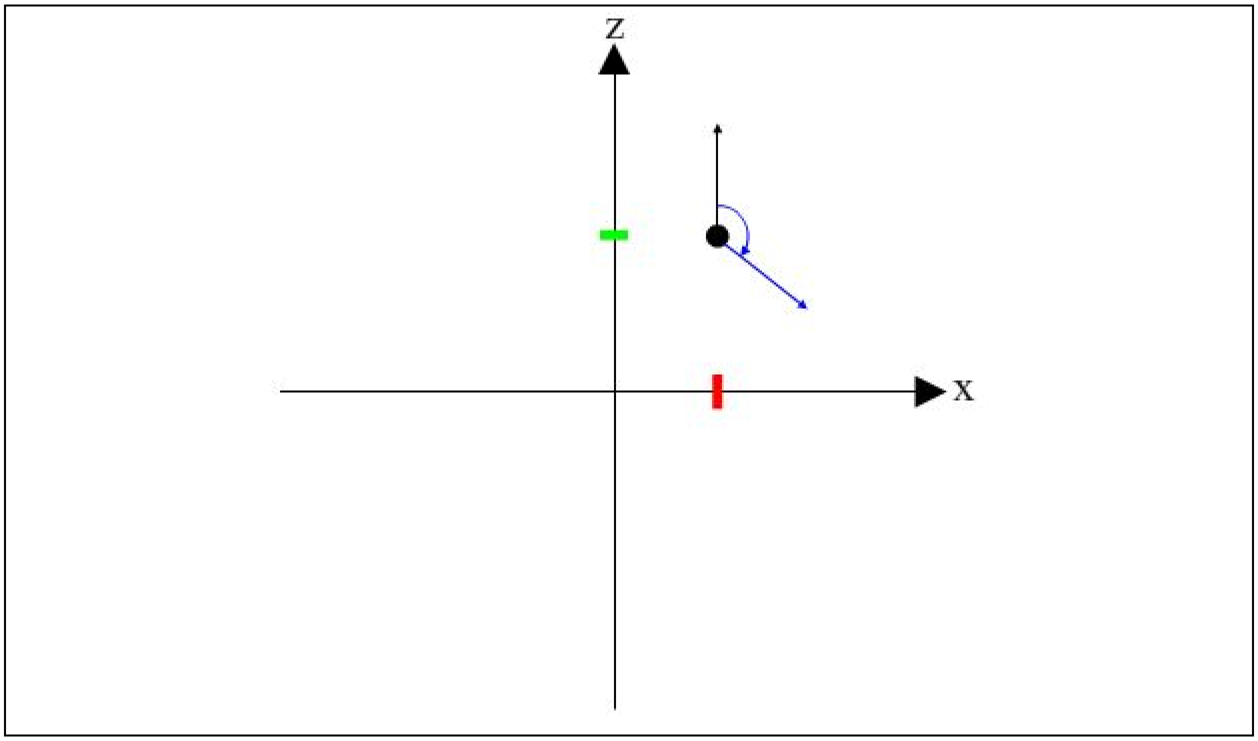

***Figure 8**: The three coordinates representing the user's virtual position. Red and green indicate x and z position while blue indicates the user's angle. The dot represents the user.*

In all cases, however, regardless of which button is pressed or what the user's angle is, the speed of motion from one button press is the same. This is accomplished by computing the speed of motion in the x and z directions independently in terms of trigonometric functions of the user's angle. The sine and cosine are computed in a "trig" module, which is essentially a lookup table mapping one of 512 possible angle values to pairs of binary outputs ranging from negative 64 to 64. We created this lookup table with a python script. As with the view angle computation, the position values are recomputed once per frame at a rate of 60Hz.

Audio (Christine)

The audio module uses the signals from the UI module to determine whether the user is pressing any of the buttons controlling player movement (button 2, button 3, "up", "down", "left", or "right"). The signal representing each button is asserted whenever the button is pressed, so if any of these signals are asserted, the user is moving in the virtual world using the buttons. This Verilog module triggers a square wave with a period of about 0.8 seconds whenever any of these signals is asserted. The square wave is assigned as a digital output to one of the FPGA pins. The output is then attached to a 20-megaohm potentiometer so that the amplitude of the square wave (i.e. the volume of the sound) can be controlled by the user. The resulting voltage is soldered to the left and right pins of an audio jack connected to the user's headphones. As a result, the user hears a periodic ticking noise which simulates the sound of footsteps.

The ticking noise does not actually come from the pitch corresponding to the low frequency of the square wave. The square wave's period is so long that the user can actually hear the wave jump from a high voltage to a low voltage and vice versa; more specifically, the jump is heard as a sudden tick. So the periodic ticking is just the periodic voltage jumps of the square wave.

Originally, we wanted to have the footstep sound be as low as 150 Hz, to be played for a quarter of a second at a time, with pockets of silence between steps, so long as the user was moving about in the virtual world. However, the sound was not as aesthetically pleasing as we had anticipated and did not resemble footsteps at all. While experimenting with lower and lower frequencies, we stumbled upon the square wave that generated the ticking sound. Realizing that this was the effect we wanted, we modified the Verilog module to remove the pockets of silence because the square wave's period was so long that we did not need the additional silence to create the footstep pulse.

Graphics Engine (Mikhail)

Module Overview

The graphics engine consists of a set of modules whose overall purpose is to put the virtual world on screen. Figure 9 is a photo of our actual project in action, with the screen showing a tetrahedron floating above several floor tiles.

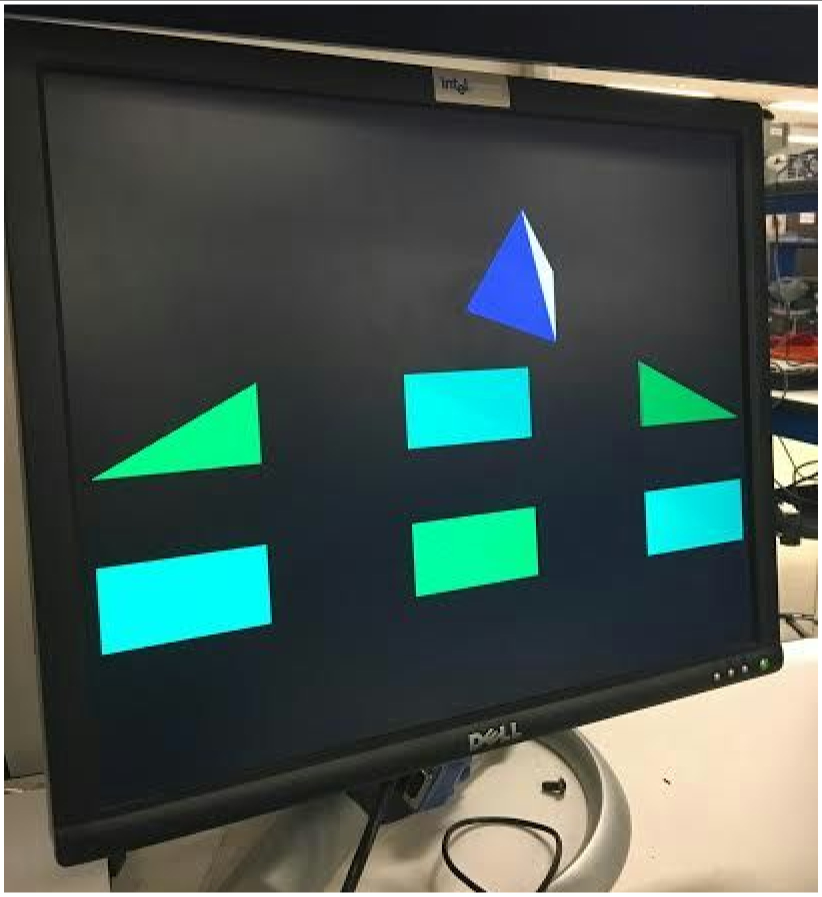

***Figure 9**: An example of the graphics engine in action.*

The graphics engine consists of two parts. The first is the VGA module from the pong-game lab, which is used to generate the necessary signals at the correct timing for 1024-by-768 resolution VGA.

The second part is mainly used to generate the color values for the pixels, though it also delays the VGA signals appropriately to account for the pipelining delay. The function of this second module is described in Figure 10.

The delayed VGA signals are easy to generate. The only difficulty comes from the color computation.

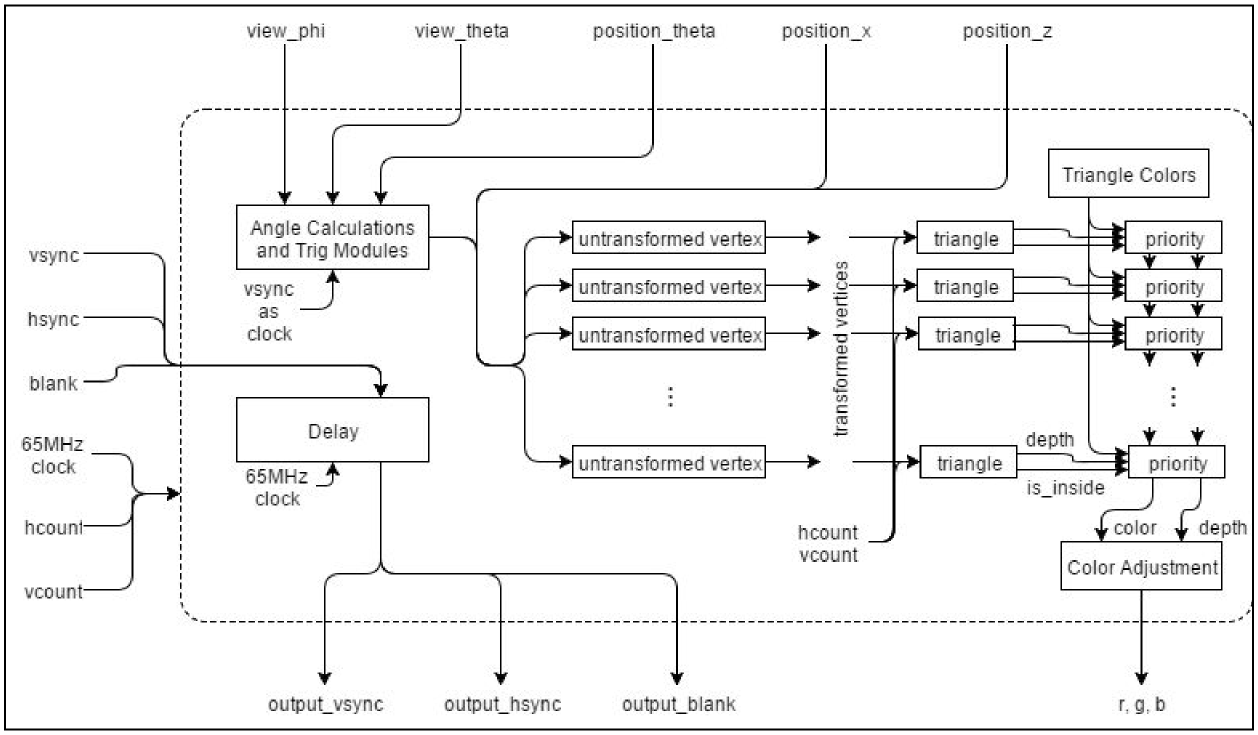

Figure 10: *A block diagram overviewing the details of the graphics engine.*

The first step in this computation is to compute the relevant angles and compute their sines and cosines. In particular, we take as input one phi angle and two theta angles. The phi angle is view_phi: the angle outputted by the view angle computation module that corresponds to pitch or vertical head motions. Similarly, view_theta is the angle outputted by the view angle computation module that corresponds to yaw or horizontal head motions. Finally, position_theta is the angle outputted by the virtual player position FSM; note that this angle is an angle measured in the x-z plane, the same plane as view_theta. From these values, we compute overall theta and phi angles: phi is set to view_phi - 25 (to create a "view a little bit from the top" as the default when the player is not moving their head up or down) and theta is set to view_theta + position_theta. After that, the sines and cosines of these angles are computed with two more instances of the trig module. Note that this is all done with a frequency of 60Hz: these calculations start and end during the vsync pulse leading up to each new frame.

These angles are used to rotate the contents of the entire world. As a result, we see that in some sense, the user's head moves independently from their feet: the player can move forward/back and side-to-side with the buttons but the direction moved in the world is not affected by any "looking around" with the gyroscope despite the change in view on screen.

The coordinates of the vertices of the triangles making up the world are hardcoded in the verilog. Each of these coordinates passes through a transform module. The transform module first subtracts the player's position (position_x and position_z) from the point's coordinates, then rotates the resulting point an angle of theta around the y-axis, next rotates the resulting point an angle of phi around the x-axis, and finally translates the point so that the range of x and y values corresponds to the range of pixels on screen. The rotations are accomplished using a rotation module, which takes sine and cosine of an angle and two coordinates as input and outputs a new version of those coordinates rotated by the given angle. This is all accomplished during the vsync pulse as well, with the output remaining constant since the input values remain constant throughout a single frame. Just in case, however, these transformed coordinates are pipelined with vsync as the clock so that they are always available immediately upon vsync going high.

The triangles, each consisting of three transformed vertices, compute whether a given hcount/vcount pair is inside or outside the view of the triangle. An additional output indicates the depth of the point on the triangle into the screen if the point is in fact inside the view of the triangle. If the point is not positioned inside the screen (if it has negative depth), it is not considered to be inside the triangle. Checking whether a point is in the view of the triangle corresponds to the problem of checking whether the point is inside the flat version of the triangle (where the z coordinate is ignored). This, in turn, can be accomplished with the intersection of three half planes, one for each edge of the triangle. Computing whether a point is in a half plane requires a simple cross product calculation, which itself requires two costly multiplications (which we pipelined).

The triangles all compute in parallel, but the next step puts these triangles in series. The triangles are "prioritized:" the triangle whose pixel has smallest depth into the screen (provided the pixel is in the triangle) is selected. This is done with a sequence of priority modules. Each module takes as input an old color, an old depth (the best color and depth "so far"), a triangle color, a triangle depth, and a boolean indicating whether the point in question is in the triangle. The output is a new color and new depth. This output simply mirrors the old color and depth unless the triangle under consideration has smaller depth and the point is in the triangle; if that occurs, the new color and depth update to those of the triangle in question. At the end of this series process is just a single output pair indicating the color of the closest pixel and the depth at which this color lies. The first priority module is given the color black and a depth that is larger than any possible depth as old color and old depth inputs.

Finally, the depth value is used to compute a factor indicating how little the color should fade: the closer to the screen, the more bright the color becomes, and the further from the screen, the more dark the color becomes. This is simply accomplished with a multiplication which scales far colors towards zero (black) and barely scales close colors at all. Note that this scaling must be accomplished independently in each color component: red, green, and blue.

Further Notes

One thing to note is that the depth computation was mathematically complicated and ended up being too expensive since it had to be done for each triangle. We approximated the depths of pixels along the triangle by simply using the average of the depths of the vertices of the triangle. This almost always yields the correct depth-ordering of the triangles, with failure only when triangles are too close together without sharing two vertices. Since we could simply build a world without that situation, this approximation saved us a lot of effort.

This design is a perfectly functional, but very inefficient way of doing things. Since there is hardware associated with every point and triangle, the system takes a very long time to compile and can support only relatively small worlds. If we were to do this again, we would implement the graphics module in a different way, using two frame buffers and a memory containing the triangles; the triangles would then be processed in series for the next frame while the current frame would repeatedly play from the frame buffer.

This particular part of the project used up a majority of the time that one of spent working on the project. A background with graphics made this possible without first writing a java or python implementation. For anyone inexperienced with graphics projects, we recommend programming a similar graphics system in a computer programming language first before attempting the task in verilog.

Reflection/Conclusion

Overall, we think this project was a success. Despite a few false starts for some of the components, we were able to put together a product which achieved almost all of the goals we set out to achieve. It was an interesting experience, and we both learned a lot. The following are some of the biggest take-away points from the different sections of the project. These are not necessarily general principles; rather they are specific facts that we would have wanted to know about this project before starting.

For the gyroscope interface, the biggest lesson learned was to use an Arduino. The L3GD20 is highly compatible with Arduino but less obviously usable with just an FPGA. Simply using an Arduino would have saved us a lot of time. In addition, we found the datasheet to be a bit unclear, and in the future we will probably spend more time examining a chip’s datasheet before buying the chip.

For the view angle computation, we found that simplifying assumptions are good. We were able to make the math much more doable simply by making a couple of very reasonable assumptions, and we believe that the system we came up with is actually a generally more robust system than the initially proposed design. In addition, we realized that we had a mistaken understanding of how a gyroscope works, which is something we should have figured out before buying the chip.

For the user interface, we would have liked to have had a better idea of how game console controllers work. Ultimately, we were unable to figure it out on time, so any hint of how to accomplish a game controller UI would have been welcome.

The virtual player position module worked pretty much as expected. The only major change was the removal of the configuration step, which has more to do with the view angle computation than it does with the virtual player position module.

The audio module would have been easier to do with a better understanding of possible sound patterns. The ticking sound we finally settled on was a happy accident, and was not at all the alternation between sound and silence that we expected to use.

Finally, the graphics module was implemented in an inefficient manner. The way we did it works, but does not scale to large worlds. Using a memory to store triangle data and using frame buffers rather than computing in real time would have been a better design decision. In addition, we would recommend budgeting more time for graphics than would normally be expected. Even with previous experience on the topic, it's an unfortunately difficult task.

In addition to these module-specific notes, we also found that near the deadline we were generally wishing for more time. Starting things earlier and spending more time doing instead of planning would have been generally good advice. Overall though, our virtual world project came together differently from how we originally planned but was ultimately still a success.

Appendix A: Verilog

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////
/////////
// Company:
// Engineer:
//
// Create Date:    16:40:00 11/30/2015
// Design Name:
// Module Name:    gyroscope
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////
/////////

// This module plays a low-frequency square wave to simulate footsteps
when triggered

module audio(
      input reset,
      input trigger,
      input clk,
      output sound
      );

      // counting period, how many clk cycles until we invert the square
wave voltage
      parameter PERIOD = 16000000;

      // register for current status of square wave
      reg square_wave;
      initial square_wave = 0;

      // counter
      reg [24:0] status_count;
      initial status_count = 0;

      always @ (posedge clk) begin

            // user or power reset -> reset everything to 0
            if (reset) begin
                  status_count <= 0;
                  square_wave <= 0;
            end
            // user is walking around in the world, so trigger is
asserted
            else if (trigger) begin

                  // invert square wave voltage since we have reached the
counting threshold,
                        // and start the count over
                  if (status_count >= PERIOD) begin
                        square_wave <= ~square_wave;
                        status_count <= 0;
                  end

                  // otherwise increment the counter
                  else begin
                        status_count <= status_count + 1;
                  end
            end
      end

      // update output
      assign sound = square_wave;

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company: 
// Engineer: 
//
// Create Date:    18:10:02 12/06/2015 
// Design Name: 
// Module Name:    clock_divider 
// Project Name: 
// Target Devices: 
// Tool versions: 
// Description: 
//
// Dependencies: 
//
// Revision: 
// Revision 0.01 - File Created
// Additional Comments: 
//
////////////////////////////////////////////////////////////////////////////////
//

// This module takes the regular 27MHz (~37 ns period) clock supplied by the
FPGA
      // and stretches the period to 75 us

module clock_divider(
       input reset,
    input clk,
    output reg div_clk
    );

       // counter
       reg [31:0] count;

       // counting period, how long between switching between high and low
       parameter PERIOD = 1012;

       always @ (posedge clk) begin

            // user or system startup reset -> reset the count and the clock
             if (reset) begin
                  count <= 0;
                  div_clk <= 0;
             end

            // counter has met threshold -> reset the count and invert the
current
                  // clock state
          if (count == PERIOD) begin
                div_clk <= ~div_clk;
                   count <= 0;
             end

            // otherwise increment the counter
             else begin
                count <= count + 1;
             end
       end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//

//the virtual player position FSM
module FSM(
            input reset,                                            // global
reset
            input clk,                                              // 60Hz clock
(i.e. vsync)
            output signed [13:0] position_x, position_z,          // user's
virtual coordinates
            output reg [8:0] position_theta,                       // including
angle
            input up, down, right, left, one, two, three, zero    // input
buttons
    );


      reg signed [19:0] full_position_x, full_position_z;

      wire signed [7:0] sin, cos;

      // compute sin and cos
      trig player_trig(.inp(position_theta[8:0]), .sin(sin[7:0]),
.cos(cos[7:0]));

      // compute x and z deltas according to buttons and trig functions
      wire signed [19:0] delta_x, delta_z;
      assign delta_x = $signed({up ? sin : 8'b0, 3'b0})
                        - $signed({down ? sin : 8'b0, 3'b0})
                                          + $signed({right ? cos : 8'b0, 2'b0})
                                          - $signed({left ? cos : 8'b0, 2'b0});
      assign delta_z = $signed({up ? cos : 8'b0, 3'b0})
                        - $signed({down ? cos : 8'b0, 3'b0})
                                          - $signed({right ? sin : 8'b0, 2'b0})
                                          + $signed({left ? sin : 8'b0, 2'b0});

      always @(posedge clk) begin
            if(reset) begin
                  full_position_x <= 0;
                  full_position_z <= 0;
                  position_theta <= 0;
            end
            else begin
                  // update angle
                  if(three == two)begin
                        position_theta <= position_theta;
                  end
                  else if (two) begin
                        position_theta <= position_theta - 1;
                  end
                  else begin
                        position_theta <= position_theta + 1;
                  end

                  // update position
                  full_position_x <= full_position_x + delta_x;
                  full_position_z <= full_position_z + delta_z;
            end
      end

      assign position_x[13:0] = full_position_x[19:6];
      assign position_z[13:0] = full_position_z[19:6];
```

```
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date:    16:40:00 11/30/2015
// Design Name:
// Module Name:    gyroscope
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//

// This module asserts different signals according to which FPGA button is being
      // pressed.  I made a module for this instead of just using the debounced
      // buttons because I wanted to make sure that only one translational
movement
      // could be performed at a time.

module game_controller(
      input reset,
      input clk,
      input left_press,
      input right_press,
      input up_press,
      input down_press,
      output left,
      output right,
      output up,
      output down
      );

      // register indicating status of left button
      reg left_reg;
      initial left_reg = 0;

      // register indicating status of right button
      reg right_reg;
      initial right_reg = 0;

      // register indicating status of up button
      reg up_reg;
      initial up_reg = 0;

      // register indicating status of down button
      reg down_reg;
      initial down_reg = 0;

      always @ (posedge clk) begin

            // user or system reset on start-up -> de-assert all button signals
            if (reset) begin
                  left_reg <= 0;
                  right_reg <= 0;
                  up_reg <= 0;
                  down_reg <= 0;
            end

            // left button was pressed, assert the left signal
            else if (left_press) begin
                  left_reg <= 1;
                  right_reg <= 0;
                  up_reg <= 0;
                  down_reg <= 0;
            end

            // right button was pressed, assert the right signal
            else if (right_press) begin
                  left_reg <= 0;
                  right_reg <= 1;
                  up_reg <= 0;
                  down_reg <= 0;
            end

            // up button was pressed, assert the up signal
            else if (up_press) begin
                  left_reg <= 0;
                  right_reg <= 0;
                  up_reg <= 1;
                  down_reg <= 0;
            end

            // down button was pressed, assert the down signal
            else if (down_press) begin
                  left_reg <= 0;
                  right_reg <= 0;
                  up_reg <= 0;
                  down_reg <= 1;
            end

            // otherwise, de-assert all signals so that they only stay
                  // active when the corr. button is pressed
            else begin
                  left_reg <= 0;
                  right_reg <= 0;
                  up_reg <= 0;
                  down_reg <= 0;
            end
      end

      // assign reg values to corresponding outputs
      assign left = left_reg;
      assign right = right_reg;
      assign up = up_reg;
      assign down = down_reg;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// graphicsModule
//
////////////////////////////////////////////////////////////////////////////////

module graphicsModule (
   input vclock,  // 65MHz clock
   input reset,         // 1 to initialize module
   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 signed [13:0] position_x, position_z,               // input user
coordinates
   input [8:0] position_theta, camera_theta, camera_phi,     // input angles

   output reg phsync,   // delayed horizontal sync
   output reg pvsync,   // delayed vertical sync
   output reg pblank,   // delayed blanking
   output [23:0] pixel  // delayed pixel  // r=23:16, g=15:8, b=7:0
   );

      parameter num_points = 40; // the number of points used
      parameter num_triangles = 22; // the number of triangles used

      //untransformed points and transformed points arrays
      wire [41:0] untransformed_points[num_points - 1:0],
transformed_points[num_points - 1:0];
      //triangle vertex arrays
      wire [41:0] triangle_pt1[num_triangles - 1:0], triangle_pt2[num_triangles
- 1:0], triangle_pt3[num_triangles - 1:0];
      //triangle color array and color array for priority selection of closest
triangle
      wire [23:0] triangle_color[num_triangles - 1:0], col[num_triangles:0];
      //depth array for priority selection of closest triangle
      wire [15:0] dpth[num_triangles:0];

      //synchronized versions of the inputs
      reg signed [13:0] position_x_s, position_z_s;
      reg [8:0] position_theta_s, camera_theta_s, camera_phi_s;
      always @(posedge vsync) begin
            position_x_s <= position_x;
            position_z_s <= position_z;
            position_theta_s <= position_theta;
            camera_theta_s <= camera_theta;
            camera_phi_s <= camera_phi;
      end

      //theta and phi computations
      wire [8:0] theta, phi;
      assign theta = $signed(0) - position_theta_s - camera_theta_s;
      assign phi = $signed(0) - camera_phi_s - $signed(25);
      wire [7:0] sin_theta, cos_theta, sin_phi, cos_phi;
      trig theta_trig(.inp(theta), .sin(sin_theta), .cos(cos_theta));
      trig phi_trig(.inp(phi), .sin(sin_phi), .cos(cos_phi));

      reg [2:0] hsync_d, vsync_d, blank_d;
      reg [7:0] alpha;
      reg [15:0] red, green, blue;
      assign pixel = {red[15:8], green[15:8], blue[15:8]};
```

```
	always @(posedge vclock) begin
		// delay VGA signals
		{phsync, hsync_d[2:0]} <= {hsync_d[2:0], hsync};
		{pvsync, vsync_d[2:0]} <= {vsync_d[2:0], vsync};
		{pblank, blank_d[2:0]} <= {blank_d[2:0], blank};

		// compute final shading according to pedpth
		alpha <= |(dpth[num_triangles][15:12]) ? 8'b0 :
~(dpth[num_triangles][11:4]);
		red <= alpha * col[num_triangles][23:16];
		green <= alpha * col[num_triangles][15:8];
		blue <= alpha * col[num_triangles][7:0];
	end

	// coordinates hardcoded
	assign untransformed_points[0][41:28] = 0;
	assign untransformed_points[0][27:14] = -50;
	assign untransformed_points[0][13:0] = 200;
	assign untransformed_points[1][41:28] = 150;
	assign untransformed_points[1][27:14] = -50;
	assign untransformed_points[1][13:0] = 100;
	assign untransformed_points[2][41:28] = 150;
	assign untransformed_points[2][27:14] = -50;
	assign untransformed_points[2][13:0] = 300;
	assign untransformed_points[3][41:28] = 100;
	assign untransformed_points[3][27:14] = -200;
	assign untransformed_points[3][13:0] = 200;
	assign untransformed_points[4][41:28] = -100;
	assign untransformed_points[4][27:14] = 200;
	assign untransformed_points[4][13:0] = -100;
	assign untransformed_points[5][41:28] = 100;
	assign untransformed_points[5][27:14] = 200;
	assign untransformed_points[5][13:0] = -100;
	assign untransformed_points[6][41:28] = -100;
	assign untransformed_points[6][27:14] = 200;
	assign untransformed_points[6][13:0] = 100;
	assign untransformed_points[7][41:28] = 100;
	assign untransformed_points[7][27:14] = 200;
	assign untransformed_points[7][13:0] = 100;
	assign untransformed_points[8][41:28] = 300;
	assign untransformed_points[8][27:14] = 200;
	assign untransformed_points[8][13:0] = -100;
	assign untransformed_points[9][41:28] = 500;
	assign untransformed_points[9][27:14] = 200;
	assign untransformed_points[9][13:0] = -100;
	assign untransformed_points[10][41:28] = 300;
	assign untransformed_points[10][27:14] = 200;
	assign untransformed_points[10][13:0] = 100;
	assign untransformed_points[11][41:28] = 500;
	assign untransformed_points[11][27:14] = 200;
	assign untransformed_points[11][13:0] = 100;
	assign untransformed_points[12][41:28] = -100;
	assign untransformed_points[12][27:14] = 200;
	assign untransformed_points[12][13:0] = 300;
	assign untransformed_points[13][41:28] = 100;
	assign untransformed_points[13][27:14] = 200;
	assign untransformed_points[13][13:0] = 300;
	assign untransformed_points[14][41:28] = -100;
	assign untransformed_points[14][27:14] = 200;
	assign untransformed_points[14][13:0] = 500;
	assign untransformed_points[15][41:28] = 100;
	assign untransformed_points[15][27:14] = 200;
	assign untransformed_points[15][13:0] = 500;
assign untransformed_points[16][41:28] = -300;
assign untransformed_points[16][27:14] = 200;
assign untransformed_points[16][13:0] = -100;
assign untransformed_points[17][41:28] = -500;
assign untransformed_points[17][27:14] = 200;
assign untransformed_points[17][13:0] = -100;
assign untransformed_points[18][41:28] = -300;
assign untransformed_points[18][27:14] = 200;
assign untransformed_points[18][13:0] = 100;
assign untransformed_points[19][41:28] = -500;
assign untransformed_points[19][27:14] = 200;
assign untransformed_points[19][13:0] = 100;
assign untransformed_points[20][41:28] = -100;
assign untransformed_points[20][27:14] = 200;
assign untransformed_points[20][13:0] = -300;
assign untransformed_points[21][41:28] = 100;
assign untransformed_points[21][27:14] = 200;
assign untransformed_points[21][13:0] = -300;
assign untransformed_points[22][41:28] = -100;
assign untransformed_points[22][27:14] = 200;
assign untransformed_points[22][13:0] = -500;
assign untransformed_points[23][41:28] = 100;
assign untransformed_points[23][27:14] = 200;
assign untransformed_points[23][13:0] = -500;
assign untransformed_points[24][41:28] = 300;
assign untransformed_points[24][27:14] = 200;
assign untransformed_points[24][13:0] = 300;
assign untransformed_points[25][41:28] = 500;
assign untransformed_points[25][27:14] = 200;
assign untransformed_points[25][13:0] = 300;
assign untransformed_points[26][41:28] = 300;
assign untransformed_points[26][27:14] = 200;
assign untransformed_points[26][13:0] = 500;
assign untransformed_points[27][41:28] = 300;
assign untransformed_points[27][27:14] = 200;
assign untransformed_points[27][13:0] = 500;
assign untransformed_points[28][41:28] = -300;
assign untransformed_points[28][27:14] = 200;
assign untransformed_points[28][13:0] = 300;
assign untransformed_points[29][41:28] = -500;
assign untransformed_points[29][27:14] = 200;
assign untransformed_points[29][13:0] = 300;
assign untransformed_points[30][41:28] = -300;
assign untransformed_points[30][27:14] = 200;
assign untransformed_points[30][13:0] = 500;
assign untransformed_points[31][41:28] = -300;
assign untransformed_points[31][27:14] = 200;
assign untransformed_points[31][13:0] = 500;
assign untransformed_points[32][41:28] = 300;
assign untransformed_points[32][27:14] = 200;
assign untransformed_points[32][13:0] = -300;
assign untransformed_points[33][41:28] = 500;
assign untransformed_points[33][27:14] = 200;
assign untransformed_points[33][13:0] = -300;
assign untransformed_points[34][41:28] = 300;
assign untransformed_points[34][27:14] = 200;
assign untransformed_points[34][13:0] = -500;
assign untransformed_points[35][41:28] = 300;
assign untransformed_points[35][27:14] = 200;
assign untransformed_points[35][13:0] = -500;
assign untransformed_points[36][41:28] = -300;
assign untransformed_points[36][27:14] = 200;
assign untransformed_points[36][13:0] = -300;
assign untransformed_points[37][41:28] = -500;
```

```
      assign untransformed_points[37][27:14] = 200;
      assign untransformed_points[37][13:0] = -300;
      assign untransformed_points[38][41:28] = -300;
      assign untransformed_points[38][27:14] = 200;
      assign untransformed_points[38][13:0] = -500;
      assign untransformed_points[39][41:28] = -300;
      assign untransformed_points[39][27:14] = 200;
      assign untransformed_points[39][13:0] = -500;


      // loop to transform all coordinates
      genvar i;
      generate
            for(i = 0; i < num_points; i = i + 1) begin:trans_block
                  transform_and_delay trans(
                        .sin_theta(sin_theta), .cos_theta(cos_theta),
                        .sin_phi(sin_phi), .cos_phi(cos_phi),
                        .position_x(position_x_s[13:0]),
.position_z(position_z_s[13:0]),
                        .point_in(untransformed_points[i]),
                        .vsync(vsync),
                        .point_out(transformed_points[i])
                  );
            end
      endgenerate

      // assigning the points to triangles
      assign triangle_pt1[0] = transformed_points[0];
      assign triangle_pt2[0] = transformed_points[1];
      assign triangle_pt3[0] = transformed_points[2];
      assign triangle_pt1[1] = transformed_points[0];
      assign triangle_pt2[1] = transformed_points[1];
      assign triangle_pt3[1] = transformed_points[3];
      assign triangle_pt1[2] = transformed_points[0];
      assign triangle_pt2[2] = transformed_points[2];
      assign triangle_pt3[2] = transformed_points[3];
      assign triangle_pt1[3] = transformed_points[1];
      assign triangle_pt2[3] = transformed_points[2];
      assign triangle_pt3[3] = transformed_points[3];
      assign triangle_pt1[4] = transformed_points[4];
      assign triangle_pt2[4] = transformed_points[5];
      assign triangle_pt3[4] = transformed_points[6];
      assign triangle_pt1[5] = transformed_points[5];
      assign triangle_pt2[5] = transformed_points[6];
      assign triangle_pt3[5] = transformed_points[7];
      assign triangle_pt1[6] = transformed_points[8];
      assign triangle_pt2[6] = transformed_points[9];
      assign triangle_pt3[6] = transformed_points[10];
      assign triangle_pt1[7] = transformed_points[9];
      assign triangle_pt2[7] = transformed_points[10];
      assign triangle_pt3[7] = transformed_points[11];
      assign triangle_pt1[8] = transformed_points[12];
      assign triangle_pt2[8] = transformed_points[13];
      assign triangle_pt3[8] = transformed_points[14];
      assign triangle_pt1[9] = transformed_points[13];
      assign triangle_pt2[9] = transformed_points[14];
      assign triangle_pt3[9] = transformed_points[15];
      assign triangle_pt1[10] = transformed_points[16];
      assign triangle_pt2[10] = transformed_points[17];
      assign triangle_pt3[10] = transformed_points[18];
      assign triangle_pt1[11] = transformed_points[17];
      assign triangle_pt2[11] = transformed_points[18];
      assign triangle_pt3[11] = transformed_points[19];
      assign triangle_pt1[12] = transformed_points[20];
      assign triangle_pt2[12] = transformed_points[21];
      assign triangle_pt3[12] = transformed_points[22];
      assign triangle_pt1[13] = transformed_points[21];
      assign triangle_pt2[13] = transformed_points[22];
      assign triangle_pt3[13] = transformed_points[23];
      assign triangle_pt1[14] = transformed_points[24];
      assign triangle_pt2[14] = transformed_points[25];
      assign triangle_pt3[14] = transformed_points[26];
      assign triangle_pt1[15] = transformed_points[25];
      assign triangle_pt2[15] = transformed_points[26];
      assign triangle_pt3[15] = transformed_points[27];
      assign triangle_pt1[16] = transformed_points[28];
      assign triangle_pt2[16] = transformed_points[29];
      assign triangle_pt3[16] = transformed_points[30];
      assign triangle_pt1[17] = transformed_points[29];
      assign triangle_pt2[17] = transformed_points[30];
      assign triangle_pt3[17] = transformed_points[31];
      assign triangle_pt1[18] = transformed_points[32];
      assign triangle_pt2[18] = transformed_points[33];
      assign triangle_pt3[18] = transformed_points[34];
      assign triangle_pt1[19] = transformed_points[33];
      assign triangle_pt2[19] = transformed_points[34];
      assign triangle_pt3[19] = transformed_points[35];
      assign triangle_pt1[20] = transformed_points[36];
      assign triangle_pt2[20] = transformed_points[37];
      assign triangle_pt3[20] = transformed_points[38];
      assign triangle_pt1[21] = transformed_points[37];
      assign triangle_pt2[21] = transformed_points[38];
      assign triangle_pt3[21] = transformed_points[39];

      //assigning colors to triangles
      assign triangle_color[0] = 24'hff0000;
      assign triangle_color[1] = 24'h0000ff;
      assign triangle_color[2] = 24'hffff00;
      assign triangle_color[3] = 24'hffffff;
      assign triangle_color[4] = 24'h00ff00;
      assign triangle_color[5] = 24'h00ff00;
      assign triangle_color[6] = 24'h10ff70;
      assign triangle_color[7] = 24'h10ff70;
      assign triangle_color[8] = 24'h10ff70;
      assign triangle_color[9] = 24'h10ff70;
      assign triangle_color[10] = 24'h10ff70;
      assign triangle_color[11] = 24'h10ff70;
      assign triangle_color[12] = 24'h10ff70;
      assign triangle_color[13] = 24'h10ff70;
      assign triangle_color[14] = 24'h00ff00;
      assign triangle_color[15] = 24'h00ff00;
      assign triangle_color[16] = 24'h00ff00;
      assign triangle_color[17] = 24'h00ff00;
      assign triangle_color[18] = 24'h00ff00;
      assign triangle_color[19] = 24'h00ff00;
      assign triangle_color[20] = 24'h00ff00;
      assign triangle_color[21] = 24'h00ff00;

      // initial color = black, initial depth = far
      assign col[0] = 24'h000000;
      assign dpth[0] = 16'hffff;

      // create triangles loop
      genvar j;
      generate
            for(j = 0; j < num_triangles; j = j + 1) begin:triangle_block
                  overall_triangle triangle_instance(
                        .hcount(hcount), .vcount(vcount),   .clk(vclock),
                        .prev_depth(dpth[j]), .next_depth(dpth[j+1]),
                        .prev_color(col[j]), .next_color(col[j+1]),
                        .triangle_color(triangle_color[j]),
                        .pt1(triangle_pt1[j]),
                        .pt2(triangle_pt2[j]),
                        .pt3(triangle_pt3[j])
                  );
            end
      endgenerate

endmodule

//////////////////////////////////////////////////////////////////////
//
// triangle: generate triangle on screen
//
//////////////////////////////////////////////////////////////////////
module triangle
   (input [10:0] hcount,
    input [9:0] vcount,
    input clk,
    output is_inside,
    input signed [13:0] x1, y1, z1, x2, y2, z2, x3, y3, z3,
       output [15:0] depth);

  wire [5:0] side;

  assign is_inside = (&{side[0], side[2], side[4]} |
                       &{side[1], side[3], side[5]})
                                          & ($signed(depth) > $signed(0));

  half_plane side0(.hcount(hcount), .vcount(vcount), .side(side[1:0]),
.clk(clk),
      .x1(x1[13:0]), .y1(y1[13:0]),
      .x2(x2[13:0]), .y2(y2[13:0])
  );
  half_plane side1(.hcount(hcount), .vcount(vcount), .side(side[3:2]),
.clk(clk),
      .x1(x2[13:0]), .y1(y2[13:0]),
      .x2(x3[13:0]), .y2(y3[13:0])
  );
  half_plane side2(.hcount(hcount), .vcount(vcount), .side(side[5:4]),
.clk(clk),
      .x1(x3[13:0]), .y1(y3[13:0]),
      .x2(x1[13:0]), .y2(y1[13:0])
  );

  assign depth = $signed(z1) + $signed(z2) + $signed(z3);
endmodule

//////////////////////////////////////////////////////////////////////
//
// half_plane: generate halfplane on screen
//
//////////////////////////////////////////////////////////////////////
module half_plane
   (input [10:0] hcount,
    input [9:0] vcount,
    input clk,
    output reg [1:0] side,
    input signed [13:0] x1, y1, x2, y2);

  wire signed [13:0] x3, y3, dx_1, dx_2, dy_1, dy_2;
  reg signed [30:0] prod_1, prod_2;
  assign x3 = $signed({1'b0, hcount[10:0]});
  assign y3 = $signed({1'b0, vcount[9:0]});

  assign dx_1[13:0] = $signed(x2[13:0]) - $signed(x1[13:0]);
  assign dy_1[13:0] = $signed(y2[13:0]) - $signed(y1[13:0]);
  assign dx_2[13:0] = $signed(x3[13:0]) - $signed(x1[13:0]);
  assign dy_2[13:0] = $signed(y3[13:0]) - $signed(y1[13:0]);

  always @(posedge clk) begin
    prod_1 <= $signed(dx_1[13:0]) * $signed(dy_2[13:0]);
    prod_2 <= $signed(dx_2[13:0]) * $signed(dy_1[13:0]);

    side[0] <= ($signed(prod_1) > $signed(1) + $signed(prod_2));
    side[1] <= ($signed(prod_1) + $signed(1) < $signed(prod_2));
  end
endmodule

///////////////////////////////////////////////////////////////////////
//
// transform_and_delay: transform an input point and pipeline it (delay one
frame)
//
///////////////////////////////////////////////////////////////////////
module transform_and_delay
   (input signed [7:0] sin_theta, cos_theta, sin_phi, cos_phi,
    input signed [13:0] position_x, position_z,
    input [41:0] point_in,
    input vsync,
    output reg [41:0] point_out);

      wire [41:0] tmp;

      transform t(
            .sin_theta(sin_theta), .cos_theta(cos_theta),
            .sin_phi(sin_phi), .cos_phi(cos_phi),
    .position_x(position_x), .position_z(position_z),
    .x(point_in[41:28]), .y(point_in[27:14]), .z(point_in[13:0]),
    .new_x(tmp[41:28]), .new_y(tmp[27:14]), .new_z(tmp[13:0])
      );

      always @(posedge vsync) begin
            point_out <= tmp;
      end

endmodule


///////////////////////////////////////////////////////////////////////
//
// screen_triangle: the sequential priority computation part of a triangle
//
///////////////////////////////////////////////////////////////////////
module screen_triangle
   (input in_triangle,
    input [15:0] prev_depth, triangle_depth,
    output [15:0] next_depth,
    input [23:0] prev_color, triangle_color,
    output [23:0] next_color);

      wire this_triangle;
      assign this_triangle = (triangle_depth < prev_depth) && in_triangle;

      assign next_depth = this_triangle ? triangle_depth : prev_depth;
      assign next_color = this_triangle ? triangle_color : prev_color;

endmodule


//////////////////////////////////////////////////////////////////////
//
// overall_triangle: everything you need to make a triangle
//
//////////////////////////////////////////////////////////////////////
module overall_triangle
   (input [15:0] prev_depth,
    output [15:0] next_depth,
    input [23:0] prev_color, triangle_color,
    output [23:0] next_color,
    input [10:0] hcount,
    input [9:0] vcount,
    input clk,
    input [41:0] pt1, pt2, pt3);


      wire in_triangle;
      wire [15:0] triangle_depth;

      reg in_triangle_d;
      reg [15:0] triangle_depth_d;

      triangle t(
            .hcount(hcount), .vcount(vcount),
            .is_inside(in_triangle), .depth(triangle_depth), .clk(clk),
            .x1(pt1[41:28]), .y1(pt1[27:14]), .z1(pt1[13:0]),
            .x2(pt2[41:28]), .y2(pt2[27:14]), .z2(pt2[13:0]),
            .x3(pt3[41:28]), .y3(pt3[27:14]), .z3(pt3[13:0])
      );

      screen_triangle st(.in_triangle(in_triangle_d), .prev_depth(prev_depth),
          .triangle_depth(triangle_depth_d), .next_depth(next_depth),
          .prev_color(prev_color), .triangle_color(triangle_color),
          .next_color(next_color));

  always @(posedge clk) begin
    in_triangle_d <= in_triangle;
    triangle_depth_d <= triangle_depth;
  end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
///
// view angle computation module: computes every frame (60Hz)
module gyroscope_transform
   (input signed [15:0] roll_in, pitch_in, yaw_in,
    output signed [8:0] pitch_out, yaw_out,
    input slow_clk, reset);

   reg signed [15:0] roll_in_d[3:0], pitch_in_d[3:0], yaw_in_d[3:0];
   reg signed [23:0] pitch, yaw;

   // exponentially weighted average accumulator
   wire signed [35:0] pitch_with_decay, yaw_with_decay;
   assign pitch_with_decay = pitch * $signed(13'b0111111101011);
   assign yaw_with_decay = yaw * $signed(13'b0111111101011);

   assign pitch_out = pitch[23:15];
   assign yaw_out = yaw[23:15];

   always @(posedge slow_clk) begin
         // last few readings in order to do smoothing
      roll_in_d[0] <= roll_in;
      pitch_in_d[0] <= pitch_in;
      yaw_in_d[0] <= yaw_in;
      roll_in_d[1] <= roll_in_d[0];
      pitch_in_d[1] <= pitch_in_d[0];
      yaw_in_d[1] <= yaw_in_d[0];
      roll_in_d[2] <= roll_in_d[1];
      pitch_in_d[2] <= pitch_in_d[1];
      yaw_in_d[2] <= yaw_in_d[1];
      roll_in_d[3] <= roll_in_d[2];
      pitch_in_d[3] <= pitch_in_d[2];
      yaw_in_d[3] <= yaw_in_d[2];

      // update according to output_pitch = pitch, output_yaw = yaw + roll
      if (reset) begin
         pitch <= 0;
         yaw <= 0;
      end
      else begin
         pitch <= pitch_with_decay[35:12] + $signed({pitch_in_d[0], 7'b0})

      + $signed({pitch_in_d[1], 7'b0})

      + $signed({pitch_in_d[2], 7'b0})

      + $signed({pitch_in_d[3], 7'b0});
         yaw <= yaw_with_decay[35:12] + $signed({yaw_in_d[0], 7'b0}) +
$signed({roll_in_d[0], 7'b0})
                                                                              +
$signed({yaw_in_d[1], 7'b0}) + $signed({roll_in_d[1], 7'b0})
                                                                              +
$signed({yaw_in_d[2], 7'b0}) + $signed({roll_in_d[2], 7'b0})
                                                                              +
$signed({yaw_in_d[3], 7'b0}) + $signed({roll_in_d[3], 7'b0});
      end
   end
endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module (video version - 24 bits)
//
///////////////////////////////////////////////////////////////////////////////

module debounce (input reset, clock, noisy,
                 output reg clean);

   reg [19:0] count;
   reg new;

   always @(posedge clock)
     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)
//
// 2012-Sep-15: Converted to 24bit RGB
//
// 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 labkit   (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

      // VGA Output
      /*assign vga_out_red = 8'h0;
      assign vga_out_green = 8'h0;
      assign vga_out_blue = 8'h0;
      assign vga_out_sync_b = 1'b1;
      assign vga_out_blank_b = 1'b1;
      assign vga_out_pixel_clock = 1'b0;
      assign vga_out_hsync = 1'b0;
      assign vga_out_vsync = 1'b0;*/

   // 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[6:0] = 7'h7F;
      assign led[7] = ~user3[0];
   // 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[31:8] = 24'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;


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

      // 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));

   // reset signal for 65MHz clocked modules
   // _m stands for mikhail
   wire power_on_reset_m;    // remain high for first 16 clocks
   SRL16 reset_sr2 (.D(1'b0), .CLK(clock_65mhz), .Q(power_on_reset_m),
               .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr2.INIT = 16'hFFFF;

   // ENTER button is user reset
   wire reset_m,user_reset_m;
   debounce
db1(.reset(power_on_reset_m),.clock(clock_65mhz),.noisy(~button_enter),.clean(us
er_reset_m));
   assign reset_m = user_reset_m | power_on_reset_m;

      // debounce buttons
      wire up,down,one,two,three,four;
   debounce
db2(.reset(reset_m),.clock(clock_65mhz),.noisy(~button_up),.clean(up));
   debounce
db3(.reset(reset_m),.clock(clock_65mhz),.noisy(~button_down),.clean(down));
   debounce
db4orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button_right),.clean(
right));
   debounce
db5orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button_left),.clean(l
eft));
   debounce
db6orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button0),.clean(zero)
);
   debounce
db7orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button1),.clean(one))
;
   debounce
db8orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button2),.clean(two))
;
   debounce
db9orsomething(.reset(reset_m),.clock(clock_65mhz),.noisy(~button3),.clean(three
));
```

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

      wire signed [13:0] position_x, position_z;
      wire [8:0] position_theta, camera_theta, camera_phi;
      wire signed[15:0] x_change;
   wire signed[15:0] y_change;
   wire signed[15:0] z_change;
      wire get_coords;

      // graphics module hooked up to VGA
   wire [23:0] pixel;
   wire phsync,pvsync,pblank;
   graphicsModule graphics(
            .vclock(clock_65mhz),.reset(reset_m),
            .hcount(hcount),.vcount(vcount),.hsync(hsync),.vsync(vsync),.blank(b
lank),
            .phsync(phsync),.pvsync(pvsync),.pblank(pblank),.pixel(pixel),
            .position_x(position_x[13:0]), .position_z(position_z[13:0]),
            .position_theta(position_theta[8:0]),
.camera_theta(camera_theta[8:0]),
            .camera_phi(camera_phi[8:0])
      );

      // virtual player position FSM sending position values to graphics module
      // based on button presses
   FSM fsm(
            .reset(reset_m), .clk(vsync),
            .position_x(position_x[13:0]), .position_z(position_z[13:0]),
            .position_theta(position_theta[8:0]),
            .up(up), .down(down), .right(right), .left(left),
            .one(one), .two(two), .three(three), .zero(zero)
      );

      // view angle computation sending view angles to graphics module
      // based on gyroscope readings
      gyroscope_transform gt(
            .roll_in(y_change), .pitch_in(x_change), .yaw_in(z_change),
            .pitch_out(camera_phi[8:0]), .yaw_out(camera_theta[8:0]),
            .slow_clk(vsync), .reset(reset_m)
       );

   // More VGA settings
   reg [23:0] rgb;

   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 = rgb[23:16];
   assign vga_out_green = rgb[15:8];
   assign vga_out_blue = rgb[7:0];
   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;

   // Reset Button Debounce for 27MHz clocked modules
   wire power_on_reset;
   wire user_reset;
   wire reset;
   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));

   debounce db_reset(.reset(power_on_reset), .clock(clock_27mhz),
.noisy(~button_enter),
      .clean(user_reset));

   assign reset = user_reset | power_on_reset;

   /////////////////////////////////////////////////////////////////////////////
///////
   // Divider for 75us Clock
   /////////////////////////////////////////////////////////////////////////////
///////

   wire clock_75us;
   clock_divider sampler(.reset(reset), .clk(clock_27mhz),
.div_clk(clock_75us));

   /////////////////////////////////////////////////////////////////////////////
///////
   // Gyroscope Read Modules
   /////////////////////////////////////////////////////////////////////////////
///////

      assign user3[5] = get_coords;

   wire read_XL_done;
   wire read_XH_done;
   wire read_YL_done;
   wire read_YH_done;
   wire read_ZL_done;
   wire read_ZH_done;

   //convert serial external inputs into parallel bits stored in x/y/z_change
registers
   //these are used above for the view angle computation

   read gyroscope_read_XL(.reset(reset), .data_stream(user3[0]),
.clk_75us(clock_75us),
      .data_out(x_change[7:0]), .done(read_XL_done));

   read gyroscope_read_XH(.reset(reset), .data_stream(user3[1]),
.clk_75us(clock_75us),
      .data_out(x_change[15:8]), .done(read_XH_done));

   read gyroscope_read_YL(.reset(reset), .data_stream(user3[2]),
.clk_75us(clock_75us),
      .data_out(y_change[7:0]), .done(read_YL_done));

   read gyroscope_read_YH(.reset(reset), .data_stream(user3[3]),
.clk_75us(clock_75us),
      .data_out(y_change[15:8]), .done(read_YH_done));
   read gyroscope_read_ZL(.reset(reset), .data_stream(user3[6]),
.clk_75us(clock_75us),
      .data_out(z_change[7:0]), .done(read_ZL_done));

   read gyroscope_read_ZH(.reset(reset), .data_stream(user3[7]),
.clk_75us(clock_75us),
      .data_out(z_change[15:8]), .done(read_ZH_done));

      // request a reading every frame during vsync
      assign get_coords = vsync;
      assign analyzer3_clock = clock_27mhz;
      assign analyzer3_data = {z_change[15:1], get_coords};

   wire trigger;
      assign trigger = left | right | up | down | three | two;

   audio footsteps(.reset(reset), .trigger(trigger), .clk(clock_27mhz),
.sound(user3[4]));

endmodule

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

module xvga(input vclock,
            output reg [10:0] hcount,    // pixel number on current line
            output reg [9:0] vcount,      // line number
            output reg vsync,hsync,blank);

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   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
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date:    16:40:00 11/30/2015
// Design Name:
// Module Name:    read
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//

// This module is designed to read in a serial/infrared signal from the Arduino
and convert
      // it into an 8-bit parallel signal
module read(
      input reset,
      input data_stream,
      input clk_75us,
      output signed[7:0] data_out,
      output done
      );

      // register for data output
      reg signed[7:0] data_out_reg;

      // register counting the number of bits we have read so far
      reg [3:0] bits_read;

      // register indicating whether or not we have acquired this coordinate
      reg done_reg;

      // register indicating which bits we have already learned
      reg [7:0] bits_learned;

      // register indicating that we have already detected a start signal
      reg start_found;

      // number of ones detected so far in input signal
      reg [5:0] ones;

      always @ (posedge clk_75us) begin

            // user or system-reset on startup -> reset everything to 0
            if (reset) begin
                  data_out_reg <= 0;
                  bits_read <= 0;
                  done_reg <= 0;
                  bits_learned <= 0;
                  start_found <= 0;
                  ones <= 0;
            end
```

```
// we have read all 8 bits
else if (bits_read == 8) begin

      // send learned bits to our data output
      data_out_reg <= bits_learned;

      // assert done signal (for testing)
      done_reg <= 1;

      // reset number of bits read so far to 0
      bits_read <= 0;

      // reset bits learned so far to 0
      bits_learned <= 0;

      // reset signal indicating start has been found to 0
      start_found <= 0;

      // reset number of ones detected to 0
      ones <= 0;
end

// we have found the start signal
else if ((ones >= 32) && ~start_found && ~data_stream) begin

      // assert signal indicating that we found this signal
      start_found <= 1;
      bits_read <= 0;
      bits_learned <= 0;
      ones <= 0;
      done_reg <= 0;
end

// we have found a valid 1 in the input signal
else if ((ones >= 16) && start_found && ~data_stream) begin

      // shift a 1 into the MSB of our learned bits
      bits_learned[7] <= 1;

      // shift all the other bits right by 1
      bits_learned[6:0] <= bits_learned[7:1];

      // increment number of bits read
      bits_read <= bits_read + 1;

      // reset counter for number of ones detected
      ones <= 0;
end

// we have found a valid 0 in the input signal
else if ((ones >= 8) && start_found && ~data_stream) begin

      // shift a 0 into the MSB of our learned bits
      bits_learned[7] <= 0;

      // shift all the other bits right by 1
      bits_learned[6:0] <= bits_learned[7:1];

      // increment number of bits read
      bits_read <= bits_read + 1;

      // reset counter for number of ones detected
      ones <= 0;
end
```

```
            else begin

                  // we have sampled a one in the signal
                  if (data_stream) begin

                        // increment the number of ones detected, keep
everything else the same
                        ones <= ones + 1;
                        bits_learned <= bits_learned;
                        bits_read <= bits_read;
                  end

                  // do nothing
                  else begin
                        ones <= ones;
                        bits_learned <= bits_learned;
                        bits_read <= bits_read;
                  end
            end
      end

      // assign data and done register values as outputs
      assign done = done_reg;
      assign data_out = data_out_reg;


endmodule
```

```
// rotation module rotates coordinates a and b to new_a and new_b given angle
info
module rotation
   (input signed [7:0] sin_theta, cos_theta,
    input signed [11:0] a, b,
    output signed [11:0] new_a, new_b);

   wire signed [17:0] asin_theta_full, bsin_theta_full, acos_theta_full,
bcos_theta_full;

   assign asin_theta_full = a * sin_theta;
   assign bsin_theta_full = b * sin_theta;
   assign acos_theta_full = a * cos_theta;
   assign bcos_theta_full = b * cos_theta;

   assign new_a = acos_theta_full[17:6] + bsin_theta_full[17:6];
   assign new_b = bcos_theta_full[17:6] - asin_theta_full[17:6];
endmodule
```

```verilog
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:08:52 11/30/2015
// Design Name:   clock_divider
// Module Name:
/afs/athena.mit.edu/user/c/k/ckonicki/Desktop/6.111/Labs/final_project/test_cloc
k_divider.v
// Project Name:  final_project
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: clock_divider
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

// This module is a ModelSim text fixture for the clock divider module
module test_clock_divider;

	// Inputs
	reg clk;
	reg reset;

	// Outputs
	wire clk_div;

	// Instantiate the Unit Under Test (UUT)
	clock_divider uut (
		.clk(clk),
		.reset(reset),
		.clk_div(clk_div)
	);

	always #5 clk = !clk;

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;

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

		// Add stimulus here
		reset = 1;
		#10;
		reset = 0;

	end

endmodule
```

```
// transform a point's location
module transform
   (input signed [7:0] sin_theta, cos_theta, sin_phi, cos_phi,
    input signed [13:0] x, y, z, position_x, position_z,
    output signed [13:0] new_x, new_y, new_z);

  // first subtract away the player's position
      wire signed [13:0] x1, z1;
      assign x1[13:0] = $signed(x[13:0]) - $signed(position_x[13:0]);
      assign z1[13:0] = $signed(z[13:0]) - $signed(position_z[13:0]);

      // then rotate around the z axis
      wire signed [13:0] x2, z2;
      rotation rot_theta(
            .sin_theta(sin_theta), .cos_theta(cos_theta),
            .a(z1[13:0]), .b(x1[13:0]), .new_a(z2[13:0]), .new_b(x2[13:0])
      );

      // then rotate around the x axis
      wire signed [13:0] y3, z3;
      rotation rot_phi(
            .sin_theta(sin_phi), .cos_theta(cos_phi),
            .a(z2[13:0]), .b(y[13:0]), .new_a(z3[13:0]), .new_b(y3[13:0])
      );

      // finally translate so that the center is (512, 384), the center of the
screen,
      // instead of (0,0)
      assign new_x[13:0] = $signed(x2[13:0]) + $signed(14'd512);
      assign new_y[13:0] = $signed(y3[13:0]) + $signed(14'd384);
      assign new_z[13:0] = $signed(z3[13:0]);
endmodule
```

```
// trigonometry look up table
module trig
   (input [8:0] inp,   // input angle
    output reg signed [7:0] sin, cos);

      always @(*) begin
            case(inp[8:0])
            9'd0: {cos[7:0], sin[7:0]} = 16'b0100000000000000;
            9'd1: {cos[7:0], sin[7:0]} = 16'b0100000011111111;
            9'd2: {cos[7:0], sin[7:0]} = 16'b0100000011111110;
            9'd3: {cos[7:0], sin[7:0]} = 16'b0100000011111110;
            9'd4: {cos[7:0], sin[7:0]} = 16'b0100000011111101;
            9'd5: {cos[7:0], sin[7:0]} = 16'b0100000011111100;
            9'd6: {cos[7:0], sin[7:0]} = 16'b0100000011111011;
            9'd7: {cos[7:0], sin[7:0]} = 16'b0100000011111011;
            9'd8: {cos[7:0], sin[7:0]} = 16'b0100000011111010;
            9'd9: {cos[7:0], sin[7:0]} = 16'b0100000011111001;
            9'd10: {cos[7:0], sin[7:0]} = 16'b0100000011111000;
            9'd11: {cos[7:0], sin[7:0]} = 16'b0011111111110111;
            9'd12: {cos[7:0], sin[7:0]} = 16'b0011111111110111;
            9'd13: {cos[7:0], sin[7:0]} = 16'b0011111111110110;
            9'd14: {cos[7:0], sin[7:0]} = 16'b0011111111110101;
            9'd15: {cos[7:0], sin[7:0]} = 16'b0011111111110100;
            9'd16: {cos[7:0], sin[7:0]} = 16'b0011111111110100;
            9'd17: {cos[7:0], sin[7:0]} = 16'b0011111111110011;
            9'd18: {cos[7:0], sin[7:0]} = 16'b0011111011110010;
            9'd19: {cos[7:0], sin[7:0]} = 16'b0011111011110001;
            9'd20: {cos[7:0], sin[7:0]} = 16'b0011111011110000;
            9'd21: {cos[7:0], sin[7:0]} = 16'b0011111011110000;
            9'd22: {cos[7:0], sin[7:0]} = 16'b0011111011101111;
            9'd23: {cos[7:0], sin[7:0]} = 16'b0011110111101110;
            9'd24: {cos[7:0], sin[7:0]} = 16'b0011110111101101;
            9'd25: {cos[7:0], sin[7:0]} = 16'b0011110111101101;
            9'd26: {cos[7:0], sin[7:0]} = 16'b0011110111101100;
            9'd27: {cos[7:0], sin[7:0]} = 16'b0011110111101011;
            9'd28: {cos[7:0], sin[7:0]} = 16'b0011110011101010;
            9'd29: {cos[7:0], sin[7:0]} = 16'b0011110011101010;
            9'd30: {cos[7:0], sin[7:0]} = 16'b0011110011101001;
            9'd31: {cos[7:0], sin[7:0]} = 16'b0011101111101000;
            9'd32: {cos[7:0], sin[7:0]} = 16'b0011101111101000;
            9'd33: {cos[7:0], sin[7:0]} = 16'b0011101111100111;
            9'd34: {cos[7:0], sin[7:0]} = 16'b0011101111100110;
            9'd35: {cos[7:0], sin[7:0]} = 16'b0011101011100101;
            9'd36: {cos[7:0], sin[7:0]} = 16'b0011101011100101;
            9'd37: {cos[7:0], sin[7:0]} = 16'b0011101011100100;
            9'd38: {cos[7:0], sin[7:0]} = 16'b0011100111100011;
            9'd39: {cos[7:0], sin[7:0]} = 16'b0011100111100011;
            9'd40: {cos[7:0], sin[7:0]} = 16'b0011100011100010;
            9'd41: {cos[7:0], sin[7:0]} = 16'b0011100011100001;
            9'd42: {cos[7:0], sin[7:0]} = 16'b0011100011100000;
            9'd43: {cos[7:0], sin[7:0]} = 16'b0011011111100000;
            9'd44: {cos[7:0], sin[7:0]} = 16'b0011011111011111;
            9'd45: {cos[7:0], sin[7:0]} = 16'b0011011011011110;
            9'd46: {cos[7:0], sin[7:0]} = 16'b0011011011011110;
            9'd47: {cos[7:0], sin[7:0]} = 16'b0011011011011101;
            9'd48: {cos[7:0], sin[7:0]} = 16'b0011010111011100;
            9'd49: {cos[7:0], sin[7:0]} = 16'b0011010111011100;
            9'd50: {cos[7:0], sin[7:0]} = 16'b0011010011011011;
            9'd51: {cos[7:0], sin[7:0]} = 16'b0011010011011011;
            9'd52: {cos[7:0], sin[7:0]} = 16'b0011001111011010;
            9'd53: {cos[7:0], sin[7:0]} = 16'b0011001111011001;
            9'd54: {cos[7:0], sin[7:0]} = 16'b0011001011011001;
            9'd55: {cos[7:0], sin[7:0]} = 16'b0011001011011000;
            9'd56: {cos[7:0], sin[7:0]} = 16'b0011000111010111;
```

```
9'd57: {cos[7:0], sin[7:0]} = 16'b0011000111010111;
9'd58: {cos[7:0], sin[7:0]} = 16'b0011000011010110;
9'd59: {cos[7:0], sin[7:0]} = 16'b0011000011010110;
9'd60: {cos[7:0], sin[7:0]} = 16'b0010111111010101;
9'd61: {cos[7:0], sin[7:0]} = 16'b0010111111010100;
9'd62: {cos[7:0], sin[7:0]} = 16'b0010111011010100;
9'd63: {cos[7:0], sin[7:0]} = 16'b0010111011010011;
9'd64: {cos[7:0], sin[7:0]} = 16'b0010110111010011;
9'd65: {cos[7:0], sin[7:0]} = 16'b0010110111010010;
9'd66: {cos[7:0], sin[7:0]} = 16'b0010110011010010;
9'd67: {cos[7:0], sin[7:0]} = 16'b0010110011010001;
9'd68: {cos[7:0], sin[7:0]} = 16'b0010101111010001;
9'd69: {cos[7:0], sin[7:0]} = 16'b0010101011010000;
9'd70: {cos[7:0], sin[7:0]} = 16'b0010101011010000;
9'd71: {cos[7:0], sin[7:0]} = 16'b0010100111001111;
9'd72: {cos[7:0], sin[7:0]} = 16'b0010100111001111;
9'd73: {cos[7:0], sin[7:0]} = 16'b0010100011001110;
9'd74: {cos[7:0], sin[7:0]} = 16'b0010011111001110;
9'd75: {cos[7:0], sin[7:0]} = 16'b0010011111001101;
9'd76: {cos[7:0], sin[7:0]} = 16'b0010011011001101;
9'd77: {cos[7:0], sin[7:0]} = 16'b0010010111001100;
9'd78: {cos[7:0], sin[7:0]} = 16'b0010010111001100;
9'd79: {cos[7:0], sin[7:0]} = 16'b0010010011001011;
9'd80: {cos[7:0], sin[7:0]} = 16'b0010010011001011;
9'd81: {cos[7:0], sin[7:0]} = 16'b0010001111001010;
9'd82: {cos[7:0], sin[7:0]} = 16'b0010001011001010;
9'd83: {cos[7:0], sin[7:0]} = 16'b0010001011001010;
9'd84: {cos[7:0], sin[7:0]} = 16'b0010000111001001;
9'd85: {cos[7:0], sin[7:0]} = 16'b0010000011001001;
9'd86: {cos[7:0], sin[7:0]} = 16'b0010000011001000;
9'd87: {cos[7:0], sin[7:0]} = 16'b0001111111001000;
9'd88: {cos[7:0], sin[7:0]} = 16'b0001111011001000;
9'd89: {cos[7:0], sin[7:0]} = 16'b0001110111000111;
9'd90: {cos[7:0], sin[7:0]} = 16'b0001110111000111;
9'd91: {cos[7:0], sin[7:0]} = 16'b0001110011000110;
9'd92: {cos[7:0], sin[7:0]} = 16'b0001101111000110;
9'd93: {cos[7:0], sin[7:0]} = 16'b0001101111000110;
9'd94: {cos[7:0], sin[7:0]} = 16'b0001101011000101;
9'd95: {cos[7:0], sin[7:0]} = 16'b0001100111000101;
9'd96: {cos[7:0], sin[7:0]} = 16'b0001100011000101;
9'd97: {cos[7:0], sin[7:0]} = 16'b0001100011000101;
9'd98: {cos[7:0], sin[7:0]} = 16'b0001011111000100;
9'd99: {cos[7:0], sin[7:0]} = 16'b0001011011000100;
9'd100: {cos[7:0], sin[7:0]} = 16'b0001011011000100;
9'd101: {cos[7:0], sin[7:0]} = 16'b0001010111000011;
9'd102: {cos[7:0], sin[7:0]} = 16'b0001010011000011;
9'd103: {cos[7:0], sin[7:0]} = 16'b0001001111000011;
9'd104: {cos[7:0], sin[7:0]} = 16'b0001001111000011;
9'd105: {cos[7:0], sin[7:0]} = 16'b0001001011000011;
9'd106: {cos[7:0], sin[7:0]} = 16'b0001000111000010;
9'd107: {cos[7:0], sin[7:0]} = 16'b0001000011000010;
9'd108: {cos[7:0], sin[7:0]} = 16'b0001000011000010;
9'd109: {cos[7:0], sin[7:0]} = 16'b0000111111000010;
9'd110: {cos[7:0], sin[7:0]} = 16'b0000111011000010;
9'd111: {cos[7:0], sin[7:0]} = 16'b0000110111000001;
9'd112: {cos[7:0], sin[7:0]} = 16'b0000110011000001;
9'd113: {cos[7:0], sin[7:0]} = 16'b0000110011000001;
9'd114: {cos[7:0], sin[7:0]} = 16'b0000101111000001;
9'd115: {cos[7:0], sin[7:0]} = 16'b0000101011000001;
9'd116: {cos[7:0], sin[7:0]} = 16'b0000100111000001;
9'd117: {cos[7:0], sin[7:0]} = 16'b0000100111000001;
9'd118: {cos[7:0], sin[7:0]} = 16'b0000100011000000;
9'd119: {cos[7:0], sin[7:0]} = 16'b0000011111000000;
9'd120: {cos[7:0], sin[7:0]} = 16'b0000011011000000;
```

```
9'd121: {cos[7:0], sin[7:0]} = 16'b0000010111000000;
9'd122: {cos[7:0], sin[7:0]} = 16'b0000010111000000;
9'd123: {cos[7:0], sin[7:0]} = 16'b0000010011000000;
9'd124: {cos[7:0], sin[7:0]} = 16'b0000001111000000;
9'd125: {cos[7:0], sin[7:0]} = 16'b0000001011000000;
9'd126: {cos[7:0], sin[7:0]} = 16'b0000001011000000;
9'd127: {cos[7:0], sin[7:0]} = 16'b0000000111000000;
9'd128: {cos[7:0], sin[7:0]} = 16'b0000000011000000;
9'd129: {cos[7:0], sin[7:0]} = 16'b1111111111000000;
9'd130: {cos[7:0], sin[7:0]} = 16'b1111111011000000;
9'd131: {cos[7:0], sin[7:0]} = 16'b1111111011000000;
9'd132: {cos[7:0], sin[7:0]} = 16'b1111110111000000;
9'd133: {cos[7:0], sin[7:0]} = 16'b1111110011000000;
9'd134: {cos[7:0], sin[7:0]} = 16'b1111101111000000;
9'd135: {cos[7:0], sin[7:0]} = 16'b1111101111000000;
9'd136: {cos[7:0], sin[7:0]} = 16'b1111101011000000;
9'd137: {cos[7:0], sin[7:0]} = 16'b1111100111000000;
9'd138: {cos[7:0], sin[7:0]} = 16'b1111100011000000;
9'd139: {cos[7:0], sin[7:0]} = 16'b1111011111000001;
9'd140: {cos[7:0], sin[7:0]} = 16'b1111011111000001;
9'd141: {cos[7:0], sin[7:0]} = 16'b1111011011000001;
9'd142: {cos[7:0], sin[7:0]} = 16'b1111010111000001;
9'd143: {cos[7:0], sin[7:0]} = 16'b1111010011000001;
9'd144: {cos[7:0], sin[7:0]} = 16'b1111010011000001;
9'd145: {cos[7:0], sin[7:0]} = 16'b1111001111000001;
9'd146: {cos[7:0], sin[7:0]} = 16'b1111001011000010;
9'd147: {cos[7:0], sin[7:0]} = 16'b1111000111000010;
9'd148: {cos[7:0], sin[7:0]} = 16'b1111000011000010;
9'd149: {cos[7:0], sin[7:0]} = 16'b1111000011000010;
9'd150: {cos[7:0], sin[7:0]} = 16'b1110111111000010;
9'd151: {cos[7:0], sin[7:0]} = 16'b1110111011000011;
9'd152: {cos[7:0], sin[7:0]} = 16'b1110110111000011;
9'd153: {cos[7:0], sin[7:0]} = 16'b1110110111000011;
9'd154: {cos[7:0], sin[7:0]} = 16'b1110110011000011;
9'd155: {cos[7:0], sin[7:0]} = 16'b1110101111000011;
9'd156: {cos[7:0], sin[7:0]} = 16'b1110101011000100;
9'd157: {cos[7:0], sin[7:0]} = 16'b1110101011000100;
9'd158: {cos[7:0], sin[7:0]} = 16'b1110100111000100;
9'd159: {cos[7:0], sin[7:0]} = 16'b1110100011000101;
9'd160: {cos[7:0], sin[7:0]} = 16'b1110100011000101;
9'd161: {cos[7:0], sin[7:0]} = 16'b1110011111000101;
9'd162: {cos[7:0], sin[7:0]} = 16'b1110011011000101;
9'd163: {cos[7:0], sin[7:0]} = 16'b1110010111000110;
9'd164: {cos[7:0], sin[7:0]} = 16'b1110010111000110;
9'd165: {cos[7:0], sin[7:0]} = 16'b1110010011000110;
9'd166: {cos[7:0], sin[7:0]} = 16'b1110001111000111;
9'd167: {cos[7:0], sin[7:0]} = 16'b1110001111000111;
9'd168: {cos[7:0], sin[7:0]} = 16'b1110001011001000;
9'd169: {cos[7:0], sin[7:0]} = 16'b1110000111001000;
9'd170: {cos[7:0], sin[7:0]} = 16'b1110000011001000;
9'd171: {cos[7:0], sin[7:0]} = 16'b1110000011001001;
9'd172: {cos[7:0], sin[7:0]} = 16'b1101111111001001;
9'd173: {cos[7:0], sin[7:0]} = 16'b1101111011001010;
9'd174: {cos[7:0], sin[7:0]} = 16'b1101111011001010;
9'd175: {cos[7:0], sin[7:0]} = 16'b1101110111001010;
9'd176: {cos[7:0], sin[7:0]} = 16'b1101110011001011;
9'd177: {cos[7:0], sin[7:0]} = 16'b1101110011001011;
9'd178: {cos[7:0], sin[7:0]} = 16'b1101101111001100;
9'd179: {cos[7:0], sin[7:0]} = 16'b1101101111001100;
9'd180: {cos[7:0], sin[7:0]} = 16'b1101101011001101;
9'd181: {cos[7:0], sin[7:0]} = 16'b1101100111001101;
9'd182: {cos[7:0], sin[7:0]} = 16'b1101100111001110;
9'd183: {cos[7:0], sin[7:0]} = 16'b1101100011001110;
9'd184: {cos[7:0], sin[7:0]} = 16'b1101011111001111;
```

```
9'd185: {cos[7:0], sin[7:0]} = 16'b1101011111001111;
9'd186: {cos[7:0], sin[7:0]} = 16'b1101011011010000;
9'd187: {cos[7:0], sin[7:0]} = 16'b1101011011010000;
9'd188: {cos[7:0], sin[7:0]} = 16'b1101010111010001;
9'd189: {cos[7:0], sin[7:0]} = 16'b1101010011010001;
9'd190: {cos[7:0], sin[7:0]} = 16'b1101010011010010;
9'd191: {cos[7:0], sin[7:0]} = 16'b1101001111010010;
9'd192: {cos[7:0], sin[7:0]} = 16'b1101001111010011;
9'd193: {cos[7:0], sin[7:0]} = 16'b1101001011010011;
9'd194: {cos[7:0], sin[7:0]} = 16'b1101001011010100;
9'd195: {cos[7:0], sin[7:0]} = 16'b1101000111010100;
9'd196: {cos[7:0], sin[7:0]} = 16'b1101000111010101;
9'd197: {cos[7:0], sin[7:0]} = 16'b1101000011010110;
9'd198: {cos[7:0], sin[7:0]} = 16'b1101000011010110;
9'd199: {cos[7:0], sin[7:0]} = 16'b1100111111010111;
9'd200: {cos[7:0], sin[7:0]} = 16'b1100111111010111;
9'd201: {cos[7:0], sin[7:0]} = 16'b1100111011011000;
9'd202: {cos[7:0], sin[7:0]} = 16'b1100111011011001;
9'd203: {cos[7:0], sin[7:0]} = 16'b1100110111011001;
9'd204: {cos[7:0], sin[7:0]} = 16'b1100110111011010;
9'd205: {cos[7:0], sin[7:0]} = 16'b1100110011011011;
9'd206: {cos[7:0], sin[7:0]} = 16'b1100110011011011;
9'd207: {cos[7:0], sin[7:0]} = 16'b1100101111011100;
9'd208: {cos[7:0], sin[7:0]} = 16'b1100101111011100;
9'd209: {cos[7:0], sin[7:0]} = 16'b1100101011011101;
9'd210: {cos[7:0], sin[7:0]} = 16'b1100101011011110;
9'd211: {cos[7:0], sin[7:0]} = 16'b1100101011011110;
9'd212: {cos[7:0], sin[7:0]} = 16'b1100100111011111;
9'd213: {cos[7:0], sin[7:0]} = 16'b1100100111100000;
9'd214: {cos[7:0], sin[7:0]} = 16'b1100100011100000;
9'd215: {cos[7:0], sin[7:0]} = 16'b1100100011100001;
9'd216: {cos[7:0], sin[7:0]} = 16'b1100100011100010;
9'd217: {cos[7:0], sin[7:0]} = 16'b1100011111100011;
9'd218: {cos[7:0], sin[7:0]} = 16'b1100011111100011;
9'd219: {cos[7:0], sin[7:0]} = 16'b1100011011100100;
9'd220: {cos[7:0], sin[7:0]} = 16'b1100011011100101;
9'd221: {cos[7:0], sin[7:0]} = 16'b1100011011100101;
9'd222: {cos[7:0], sin[7:0]} = 16'b1100010111100110;
9'd223: {cos[7:0], sin[7:0]} = 16'b1100010111100111;
9'd224: {cos[7:0], sin[7:0]} = 16'b1100010111101000;
9'd225: {cos[7:0], sin[7:0]} = 16'b1100010111101000;
9'd226: {cos[7:0], sin[7:0]} = 16'b1100010011101001;
9'd227: {cos[7:0], sin[7:0]} = 16'b1100010011101010;
9'd228: {cos[7:0], sin[7:0]} = 16'b1100010011101010;
9'd229: {cos[7:0], sin[7:0]} = 16'b1100001111101011;
9'd230: {cos[7:0], sin[7:0]} = 16'b1100001111101100;
9'd231: {cos[7:0], sin[7:0]} = 16'b1100001111101101;
9'd232: {cos[7:0], sin[7:0]} = 16'b1100001111101101;
9'd233: {cos[7:0], sin[7:0]} = 16'b1100001111101110;
9'd234: {cos[7:0], sin[7:0]} = 16'b1100001011101111;
9'd235: {cos[7:0], sin[7:0]} = 16'b1100001011110000;
9'd236: {cos[7:0], sin[7:0]} = 16'b1100001011110000;
9'd237: {cos[7:0], sin[7:0]} = 16'b1100001011110001;
9'd238: {cos[7:0], sin[7:0]} = 16'b1100001011110010;
9'd239: {cos[7:0], sin[7:0]} = 16'b1100000111110011;
9'd240: {cos[7:0], sin[7:0]} = 16'b1100000111110100;
9'd241: {cos[7:0], sin[7:0]} = 16'b1100000111110100;
9'd242: {cos[7:0], sin[7:0]} = 16'b1100000111110101;
9'd243: {cos[7:0], sin[7:0]} = 16'b1100000111110110;
9'd244: {cos[7:0], sin[7:0]} = 16'b1100000111110111;
9'd245: {cos[7:0], sin[7:0]} = 16'b1100000111110111;
9'd246: {cos[7:0], sin[7:0]} = 16'b1100000011111000;
9'd247: {cos[7:0], sin[7:0]} = 16'b1100000011111001;
9'd248: {cos[7:0], sin[7:0]} = 16'b1100000011111010;
```

```
9'd249: {cos[7:0], sin[7:0]} = 16'b1100000011111011;
9'd250: {cos[7:0], sin[7:0]} = 16'b1100000011111011;
9'd251: {cos[7:0], sin[7:0]} = 16'b1100000011111100;
9'd252: {cos[7:0], sin[7:0]} = 16'b1100000011111101;
9'd253: {cos[7:0], sin[7:0]} = 16'b1100000011111110;
9'd254: {cos[7:0], sin[7:0]} = 16'b1100000011111110;
9'd255: {cos[7:0], sin[7:0]} = 16'b1100000011111111;
9'd256: {cos[7:0], sin[7:0]} = 16'b1100000000000000;
9'd257: {cos[7:0], sin[7:0]} = 16'b1100000000000001;
9'd258: {cos[7:0], sin[7:0]} = 16'b1100000000000010;
9'd259: {cos[7:0], sin[7:0]} = 16'b1100000000000010;
9'd260: {cos[7:0], sin[7:0]} = 16'b1100000000000011;
9'd261: {cos[7:0], sin[7:0]} = 16'b1100000000000100;
9'd262: {cos[7:0], sin[7:0]} = 16'b1100000000000101;
9'd263: {cos[7:0], sin[7:0]} = 16'b1100000000000101;
9'd264: {cos[7:0], sin[7:0]} = 16'b1100000000000110;
9'd265: {cos[7:0], sin[7:0]} = 16'b1100000000000111;
9'd266: {cos[7:0], sin[7:0]} = 16'b1100000000001000;
9'd267: {cos[7:0], sin[7:0]} = 16'b1100000100001001;
9'd268: {cos[7:0], sin[7:0]} = 16'b1100000100001001;
9'd269: {cos[7:0], sin[7:0]} = 16'b1100000100001010;
9'd270: {cos[7:0], sin[7:0]} = 16'b1100000100001011;
9'd271: {cos[7:0], sin[7:0]} = 16'b1100000100001100;
9'd272: {cos[7:0], sin[7:0]} = 16'b1100000100001100;
9'd273: {cos[7:0], sin[7:0]} = 16'b1100000100001101;
9'd274: {cos[7:0], sin[7:0]} = 16'b1100001000001110;
9'd275: {cos[7:0], sin[7:0]} = 16'b1100001000001111;
9'd276: {cos[7:0], sin[7:0]} = 16'b1100001000010000;
9'd277: {cos[7:0], sin[7:0]} = 16'b1100001000010000;
9'd278: {cos[7:0], sin[7:0]} = 16'b1100001000010001;
9'd279: {cos[7:0], sin[7:0]} = 16'b1100001100010010;
9'd280: {cos[7:0], sin[7:0]} = 16'b1100001100010011;
9'd281: {cos[7:0], sin[7:0]} = 16'b1100001100010011;
9'd282: {cos[7:0], sin[7:0]} = 16'b1100001100010100;
9'd283: {cos[7:0], sin[7:0]} = 16'b1100001100010101;
9'd284: {cos[7:0], sin[7:0]} = 16'b1100010000010110;
9'd285: {cos[7:0], sin[7:0]} = 16'b1100010000010110;
9'd286: {cos[7:0], sin[7:0]} = 16'b1100010000010111;
9'd287: {cos[7:0], sin[7:0]} = 16'b1100010100011000;
9'd288: {cos[7:0], sin[7:0]} = 16'b1100010100011000;
9'd289: {cos[7:0], sin[7:0]} = 16'b1100010100011001;
9'd290: {cos[7:0], sin[7:0]} = 16'b1100010100011010;
9'd291: {cos[7:0], sin[7:0]} = 16'b1100011000011011;
9'd292: {cos[7:0], sin[7:0]} = 16'b1100011000011011;
9'd293: {cos[7:0], sin[7:0]} = 16'b1100011000011100;
9'd294: {cos[7:0], sin[7:0]} = 16'b1100011100011101;
9'd295: {cos[7:0], sin[7:0]} = 16'b1100011100011101;
9'd296: {cos[7:0], sin[7:0]} = 16'b1100100000011110;
9'd297: {cos[7:0], sin[7:0]} = 16'b1100100000011111;
9'd298: {cos[7:0], sin[7:0]} = 16'b1100100000100000;
9'd299: {cos[7:0], sin[7:0]} = 16'b1100100100100000;
9'd300: {cos[7:0], sin[7:0]} = 16'b1100100100100001;
9'd301: {cos[7:0], sin[7:0]} = 16'b1100101000100010;
9'd302: {cos[7:0], sin[7:0]} = 16'b1100101000100010;
9'd303: {cos[7:0], sin[7:0]} = 16'b1100101000100011;
9'd304: {cos[7:0], sin[7:0]} = 16'b1100101100100100;
9'd305: {cos[7:0], sin[7:0]} = 16'b1100101100100100;
9'd306: {cos[7:0], sin[7:0]} = 16'b1100110000100101;
9'd307: {cos[7:0], sin[7:0]} = 16'b1100110000100101;
9'd308: {cos[7:0], sin[7:0]} = 16'b1100110100100110;
9'd309: {cos[7:0], sin[7:0]} = 16'b1100110100100111;
9'd310: {cos[7:0], sin[7:0]} = 16'b1100111000100111;
9'd311: {cos[7:0], sin[7:0]} = 16'b1100111000101000;
9'd312: {cos[7:0], sin[7:0]} = 16'b1100111100101001;
```

```
9'd313: {cos[7:0], sin[7:0]} = 16'b1100111100101001;
9'd314: {cos[7:0], sin[7:0]} = 16'b1101000000101010;
9'd315: {cos[7:0], sin[7:0]} = 16'b1101000000101010;
9'd316: {cos[7:0], sin[7:0]} = 16'b1101000100101011;
9'd317: {cos[7:0], sin[7:0]} = 16'b1101000100101100;
9'd318: {cos[7:0], sin[7:0]} = 16'b1101001000101100;
9'd319: {cos[7:0], sin[7:0]} = 16'b1101001000101101;
9'd320: {cos[7:0], sin[7:0]} = 16'b1101001100101101;
9'd321: {cos[7:0], sin[7:0]} = 16'b1101001100101110;
9'd322: {cos[7:0], sin[7:0]} = 16'b1101010000101110;
9'd323: {cos[7:0], sin[7:0]} = 16'b1101010000101111;
9'd324: {cos[7:0], sin[7:0]} = 16'b1101010100101111;
9'd325: {cos[7:0], sin[7:0]} = 16'b1101011000110000;
9'd326: {cos[7:0], sin[7:0]} = 16'b1101011000110000;
9'd327: {cos[7:0], sin[7:0]} = 16'b1101011100110001;
9'd328: {cos[7:0], sin[7:0]} = 16'b1101011100110001;
9'd329: {cos[7:0], sin[7:0]} = 16'b1101100000110010;
9'd330: {cos[7:0], sin[7:0]} = 16'b1101100100110010;
9'd331: {cos[7:0], sin[7:0]} = 16'b1101100100110011;
9'd332: {cos[7:0], sin[7:0]} = 16'b1101101000110011;
9'd333: {cos[7:0], sin[7:0]} = 16'b1101101100110100;
9'd334: {cos[7:0], sin[7:0]} = 16'b1101101100110100;
9'd335: {cos[7:0], sin[7:0]} = 16'b1101110000110101;
9'd336: {cos[7:0], sin[7:0]} = 16'b1101110000110101;
9'd337: {cos[7:0], sin[7:0]} = 16'b1101110100110110;
9'd338: {cos[7:0], sin[7:0]} = 16'b1101111000110110;
9'd339: {cos[7:0], sin[7:0]} = 16'b1101111000110110;
9'd340: {cos[7:0], sin[7:0]} = 16'b1101111100110111;
9'd341: {cos[7:0], sin[7:0]} = 16'b1110000000110111;
9'd342: {cos[7:0], sin[7:0]} = 16'b1110000000111000;
9'd343: {cos[7:0], sin[7:0]} = 16'b1110000100111000;
9'd344: {cos[7:0], sin[7:0]} = 16'b1110001000111000;
9'd345: {cos[7:0], sin[7:0]} = 16'b1110001100111001;
9'd346: {cos[7:0], sin[7:0]} = 16'b1110001100111001;
9'd347: {cos[7:0], sin[7:0]} = 16'b1110010000111010;
9'd348: {cos[7:0], sin[7:0]} = 16'b1110010100111010;
9'd349: {cos[7:0], sin[7:0]} = 16'b1110010100111010;
9'd350: {cos[7:0], sin[7:0]} = 16'b1110011000111011;
9'd351: {cos[7:0], sin[7:0]} = 16'b1110011100111011;
9'd352: {cos[7:0], sin[7:0]} = 16'b1110100000111011;
9'd353: {cos[7:0], sin[7:0]} = 16'b1110100000111011;
9'd354: {cos[7:0], sin[7:0]} = 16'b1110100100111100;
9'd355: {cos[7:0], sin[7:0]} = 16'b1110101000111100;
9'd356: {cos[7:0], sin[7:0]} = 16'b1110101000111100;
9'd357: {cos[7:0], sin[7:0]} = 16'b1110101100111101;
9'd358: {cos[7:0], sin[7:0]} = 16'b1110110000111101;
9'd359: {cos[7:0], sin[7:0]} = 16'b1110110100111101;
9'd360: {cos[7:0], sin[7:0]} = 16'b1110110100111101;
9'd361: {cos[7:0], sin[7:0]} = 16'b1110111000111101;
9'd362: {cos[7:0], sin[7:0]} = 16'b1110111100111110;
9'd363: {cos[7:0], sin[7:0]} = 16'b1111000000111110;
9'd364: {cos[7:0], sin[7:0]} = 16'b1111000000111110;
9'd365: {cos[7:0], sin[7:0]} = 16'b1111000100111110;
9'd366: {cos[7:0], sin[7:0]} = 16'b1111001000111110;
9'd367: {cos[7:0], sin[7:0]} = 16'b1111001100111111;
9'd368: {cos[7:0], sin[7:0]} = 16'b1111010000111111;
9'd369: {cos[7:0], sin[7:0]} = 16'b1111010000111111;
9'd370: {cos[7:0], sin[7:0]} = 16'b1111010100111111;
9'd371: {cos[7:0], sin[7:0]} = 16'b1111011000111111;
9'd372: {cos[7:0], sin[7:0]} = 16'b1111011100111111;
9'd373: {cos[7:0], sin[7:0]} = 16'b1111011100111111;
9'd374: {cos[7:0], sin[7:0]} = 16'b1111100001000000;
9'd375: {cos[7:0], sin[7:0]} = 16'b1111100101000000;
9'd376: {cos[7:0], sin[7:0]} = 16'b1111101001000000;
```

```
9'd377: {cos[7:0], sin[7:0]} = 16'b1111101101000000;
9'd378: {cos[7:0], sin[7:0]} = 16'b1111101101000000;
9'd379: {cos[7:0], sin[7:0]} = 16'b1111110001000000;
9'd380: {cos[7:0], sin[7:0]} = 16'b1111110101000000;
9'd381: {cos[7:0], sin[7:0]} = 16'b1111111001000000;
9'd382: {cos[7:0], sin[7:0]} = 16'b1111111001000000;
9'd383: {cos[7:0], sin[7:0]} = 16'b1111111101000000;
9'd384: {cos[7:0], sin[7:0]} = 16'b0000000001000000;
9'd385: {cos[7:0], sin[7:0]} = 16'b0000000101000000;
9'd386: {cos[7:0], sin[7:0]} = 16'b0000001001000000;
9'd387: {cos[7:0], sin[7:0]} = 16'b0000001001000000;
9'd388: {cos[7:0], sin[7:0]} = 16'b0000001101000000;
9'd389: {cos[7:0], sin[7:0]} = 16'b0000010001000000;
9'd390: {cos[7:0], sin[7:0]} = 16'b0000010101000000;
9'd391: {cos[7:0], sin[7:0]} = 16'b0000010101000000;
9'd392: {cos[7:0], sin[7:0]} = 16'b0000011001000000;
9'd393: {cos[7:0], sin[7:0]} = 16'b0000011101000000;
9'd394: {cos[7:0], sin[7:0]} = 16'b0000100001000000;
9'd395: {cos[7:0], sin[7:0]} = 16'b0000100100111111;
9'd396: {cos[7:0], sin[7:0]} = 16'b0000100100111111;
9'd397: {cos[7:0], sin[7:0]} = 16'b0000101000111111;
9'd398: {cos[7:0], sin[7:0]} = 16'b0000101100111111;
9'd399: {cos[7:0], sin[7:0]} = 16'b0000110000111111;
9'd400: {cos[7:0], sin[7:0]} = 16'b0000110000111111;
9'd401: {cos[7:0], sin[7:0]} = 16'b0000110100111111;
9'd402: {cos[7:0], sin[7:0]} = 16'b0000111000111110;
9'd403: {cos[7:0], sin[7:0]} = 16'b0000111100111110;
9'd404: {cos[7:0], sin[7:0]} = 16'b0001000000111110;
9'd405: {cos[7:0], sin[7:0]} = 16'b0001000000111110;
9'd406: {cos[7:0], sin[7:0]} = 16'b0001000100111110;
9'd407: {cos[7:0], sin[7:0]} = 16'b0001001000111101;
9'd408: {cos[7:0], sin[7:0]} = 16'b0001001100111101;
9'd409: {cos[7:0], sin[7:0]} = 16'b0001001100111101;
9'd410: {cos[7:0], sin[7:0]} = 16'b0001010000111101;
9'd411: {cos[7:0], sin[7:0]} = 16'b0001010100111101;
9'd412: {cos[7:0], sin[7:0]} = 16'b0001011000111100;
9'd413: {cos[7:0], sin[7:0]} = 16'b0001011000111100;
9'd414: {cos[7:0], sin[7:0]} = 16'b0001011100111100;
9'd415: {cos[7:0], sin[7:0]} = 16'b0001100000111011;
9'd416: {cos[7:0], sin[7:0]} = 16'b0001100000111011;
9'd417: {cos[7:0], sin[7:0]} = 16'b0001100100111011;
9'd418: {cos[7:0], sin[7:0]} = 16'b0001101000111011;
9'd419: {cos[7:0], sin[7:0]} = 16'b0001101100111010;
9'd420: {cos[7:0], sin[7:0]} = 16'b0001101100111010;
9'd421: {cos[7:0], sin[7:0]} = 16'b0001110000111010;
9'd422: {cos[7:0], sin[7:0]} = 16'b0001110100111001;
9'd423: {cos[7:0], sin[7:0]} = 16'b0001110100111001;
9'd424: {cos[7:0], sin[7:0]} = 16'b0001111000111000;
9'd425: {cos[7:0], sin[7:0]} = 16'b0001111100111000;
9'd426: {cos[7:0], sin[7:0]} = 16'b0010000000111000;
9'd427: {cos[7:0], sin[7:0]} = 16'b0010000000110111;
9'd428: {cos[7:0], sin[7:0]} = 16'b0010000100110111;
9'd429: {cos[7:0], sin[7:0]} = 16'b0010001000110110;
9'd430: {cos[7:0], sin[7:0]} = 16'b0010001000110110;
9'd431: {cos[7:0], sin[7:0]} = 16'b0010001100110110;
9'd432: {cos[7:0], sin[7:0]} = 16'b0010010000110101;
9'd433: {cos[7:0], sin[7:0]} = 16'b0010010000110101;
9'd434: {cos[7:0], sin[7:0]} = 16'b0010010100110100;
9'd435: {cos[7:0], sin[7:0]} = 16'b0010010100110100;
9'd436: {cos[7:0], sin[7:0]} = 16'b0010011000110011;
9'd437: {cos[7:0], sin[7:0]} = 16'b0010011100110011;
9'd438: {cos[7:0], sin[7:0]} = 16'b0010011100110010;
9'd439: {cos[7:0], sin[7:0]} = 16'b0010100000110010;
9'd440: {cos[7:0], sin[7:0]} = 16'b0010100100110001;
```

```
9'd441: {cos[7:0], sin[7:0]} = 16'b0010100100110001;
9'd442: {cos[7:0], sin[7:0]} = 16'b0010101000110000;
9'd443: {cos[7:0], sin[7:0]} = 16'b0010101000110000;
9'd444: {cos[7:0], sin[7:0]} = 16'b0010101100101111;
9'd445: {cos[7:0], sin[7:0]} = 16'b0010110000101111;
9'd446: {cos[7:0], sin[7:0]} = 16'b0010110000101110;
9'd447: {cos[7:0], sin[7:0]} = 16'b0010110100101110;
9'd448: {cos[7:0], sin[7:0]} = 16'b0010110100101101;
9'd449: {cos[7:0], sin[7:0]} = 16'b0010111000101101;
9'd450: {cos[7:0], sin[7:0]} = 16'b0010111000101100;
9'd451: {cos[7:0], sin[7:0]} = 16'b0010111100101100;
9'd452: {cos[7:0], sin[7:0]} = 16'b0010111100101011;
9'd453: {cos[7:0], sin[7:0]} = 16'b0011000000101010;
9'd454: {cos[7:0], sin[7:0]} = 16'b0011000000101010;
9'd455: {cos[7:0], sin[7:0]} = 16'b0011000100101001;
9'd456: {cos[7:0], sin[7:0]} = 16'b0011000100101001;
9'd457: {cos[7:0], sin[7:0]} = 16'b0011001000101000;
9'd458: {cos[7:0], sin[7:0]} = 16'b0011001000100111;
9'd459: {cos[7:0], sin[7:0]} = 16'b0011001100100111;
9'd460: {cos[7:0], sin[7:0]} = 16'b0011001100100110;
9'd461: {cos[7:0], sin[7:0]} = 16'b0011010000100101;
9'd462: {cos[7:0], sin[7:0]} = 16'b0011010000100101;
9'd463: {cos[7:0], sin[7:0]} = 16'b0011010100100100;
9'd464: {cos[7:0], sin[7:0]} = 16'b0011010100100100;
9'd465: {cos[7:0], sin[7:0]} = 16'b0011011000100011;
9'd466: {cos[7:0], sin[7:0]} = 16'b0011011000100010;
9'd467: {cos[7:0], sin[7:0]} = 16'b0011011000100010;
9'd468: {cos[7:0], sin[7:0]} = 16'b0011011100100001;
9'd469: {cos[7:0], sin[7:0]} = 16'b0011011100100000;
9'd470: {cos[7:0], sin[7:0]} = 16'b0011100000100000;
9'd471: {cos[7:0], sin[7:0]} = 16'b0011100000011111;
9'd472: {cos[7:0], sin[7:0]} = 16'b0011100000011110;
9'd473: {cos[7:0], sin[7:0]} = 16'b0011100100011101;
9'd474: {cos[7:0], sin[7:0]} = 16'b0011100100011101;
9'd475: {cos[7:0], sin[7:0]} = 16'b0011101000011100;
9'd476: {cos[7:0], sin[7:0]} = 16'b0011101000011011;
9'd477: {cos[7:0], sin[7:0]} = 16'b0011101000011011;
9'd478: {cos[7:0], sin[7:0]} = 16'b0011101100011010;
9'd479: {cos[7:0], sin[7:0]} = 16'b0011101100011001;
9'd480: {cos[7:0], sin[7:0]} = 16'b0011101100011000;
9'd481: {cos[7:0], sin[7:0]} = 16'b0011101100011000;
9'd482: {cos[7:0], sin[7:0]} = 16'b0011110000010111;
9'd483: {cos[7:0], sin[7:0]} = 16'b0011110000010110;
9'd484: {cos[7:0], sin[7:0]} = 16'b0011110000010110;
9'd485: {cos[7:0], sin[7:0]} = 16'b0011110100010101;
9'd486: {cos[7:0], sin[7:0]} = 16'b0011110100010100;
9'd487: {cos[7:0], sin[7:0]} = 16'b0011110100010011;
9'd488: {cos[7:0], sin[7:0]} = 16'b0011110100010011;
9'd489: {cos[7:0], sin[7:0]} = 16'b0011110100010010;
9'd490: {cos[7:0], sin[7:0]} = 16'b0011111000010001;
9'd491: {cos[7:0], sin[7:0]} = 16'b0011111000010000;
9'd492: {cos[7:0], sin[7:0]} = 16'b0011111000010000;
9'd493: {cos[7:0], sin[7:0]} = 16'b0011111000001111;
9'd494: {cos[7:0], sin[7:0]} = 16'b0011111000001110;
9'd495: {cos[7:0], sin[7:0]} = 16'b0011111100001101;
9'd496: {cos[7:0], sin[7:0]} = 16'b0011111100001100;
9'd497: {cos[7:0], sin[7:0]} = 16'b0011111100001100;
9'd498: {cos[7:0], sin[7:0]} = 16'b0011111100001011;
9'd499: {cos[7:0], sin[7:0]} = 16'b0011111100001010;
9'd500: {cos[7:0], sin[7:0]} = 16'b0011111100001001;
9'd501: {cos[7:0], sin[7:0]} = 16'b0011111100001001;
9'd502: {cos[7:0], sin[7:0]} = 16'b0100000000001000;
9'd503: {cos[7:0], sin[7:0]} = 16'b0100000000000111;
9'd504: {cos[7:0], sin[7:0]} = 16'b0100000000000110;
```

```
            9'd505: {cos[7:0], sin[7:0]} = 16'b0100000000000101;
            9'd506: {cos[7:0], sin[7:0]} = 16'b0100000000000101;
            9'd507: {cos[7:0], sin[7:0]} = 16'b0100000000000100;
            9'd508: {cos[7:0], sin[7:0]} = 16'b0100000000000011;
            9'd509: {cos[7:0], sin[7:0]} = 16'b0100000000000010;
            9'd510: {cos[7:0], sin[7:0]} = 16'b0100000000000010;
            9'd511: {cos[7:0], sin[7:0]} = 16'b0100000000000001;
            default: {cos[7:0], sin[7:0]} = 16'b0100000000000000;
            endcase
      end

endmodule
```

Appendix B: Arduino Code

```
/***************************************************
  This is a library for the L3GD20 and L3GD20H GYROSCOPE

  Designed specifically to work with the Adafruit L3GD20(H) Breakout
  ----> https://www.adafruit.com/products/1032

  These sensors use I2C or SPI to communicate, 2 pins (I2C)
  or 4 pins (SPI) are required to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Kevin "KTOWN" Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ****************************************************/

#include <Adafruit_L3GD20.h>

/***************************************************************************
 CONSTRUCTOR
 ***************************************************************************/

Adafruit_L3GD20::Adafruit_L3GD20(int8_t cs, int8_t miso, int8_t mosi, int8_t
clk) {
  _cs = cs;
  _miso = miso;
  _mosi = mosi;
  _clk = clk;
}

Adafruit_L3GD20::Adafruit_L3GD20(void) {
  // use i2c
  _cs = _mosi = _miso = _clk = -1;
}

bool Adafruit_L3GD20::begin(l3gd20Range_t rng, byte addr)
{
  if (_cs == -1) {
    Wire.begin();
  } else {
    pinMode(_cs, OUTPUT);
    pinMode(_clk, OUTPUT);
    pinMode(_mosi, OUTPUT);
    pinMode(_miso, INPUT);
    digitalWrite(_cs, HIGH);
  }

  address = addr;
  range = rng;

  /* Make sure we have the correct chip ID since this checks
     for correct address and that the IC is properly connected */
  uint8_t id = read8(L3GD20_REGISTER_WHO_AM_I);
  //Serial.println(id, HEX);
  if ((id != L3GD20_ID) && (id != L3GD20H_ID))
  {
    return false;
  }

  /* Set CTRL_REG1 (0x20)
   ====================================================================
   BIT  Symbol    Description                                   Default
   ---  ------    --------------------------------------------- -------
   7-6  DR1/0     Output data rate                                  00
   5-4  BW1/0     Bandwidth selection                               00
     3  PD        0 = Power-down mode, 1 = normal/sleep mode         0
     2  ZEN       Z-axis enable (0 = disabled, 1 = enabled)          1
     1  YEN       Y-axis enable (0 = disabled, 1 = enabled)          1
     0  XEN       X-axis enable (0 = disabled, 1 = enabled)          1 */

  /* Switch to normal mode and enable all three channels */
  write8(L3GD20_REGISTER_CTRL_REG1, 0x0F);
  /* ------------------------------------------------------------------ */

  /* Set CTRL_REG2 (0x21)
   ====================================================================
   BIT  Symbol    Description                                   Default
   ---  ------    --------------------------------------------- -------
   5-4  HPM1/0    High-pass filter mode selection                   00
   3-0  HPCF3..0  High-pass filter cutoff frequency selection     0000 */

  /* Nothing to do ... keep default values */
  /* ------------------------------------------------------------------ */

  /* Set CTRL_REG3 (0x22)
   ====================================================================
   BIT  Symbol    Description                                   Default
   ---  ------    --------------------------------------------- -------
     7  I1_Int1   Interrupt enable on INT1 (0=disable,1=enable)      0
     6  I1_Boot   Boot status on INT1 (0=disable,1=enable)           0
     5  H-Lactive Interrupt active config on INT1 (0=high,1=low)     0
     4  PP_OD     Push-Pull/Open-Drain (0=PP, 1=OD)                  0
     3  I2_DRDY   Data ready on DRDY/INT2 (0=disable,1=enable)       0
     2  I2_WTM    FIFO wtrmrk int on DRDY/INT2 (0=dsbl,1=enbl)       0
     1  I2_ORun   FIFO overrun int on DRDY/INT2 (0=dsbl,1=enbl)      0
     0  I2_Empty  FIFI empty int on DRDY/INT2 (0=dsbl,1=enbl)        0 */

  /* Nothing to do ... keep default values */
  /* ------------------------------------------------------------------ */

  /* Set CTRL_REG4 (0x23)
   ====================================================================
   BIT  Symbol    Description                                   Default
   ---  ------    --------------------------------------------- -------
     7  BDU       Block Data Update (0=continuous, 1=LSB/MSB)        0
     6  BLE       Big/Little-Endian (0=Data LSB, 1=Data MSB)         0
   5-4  FS1/0     Full scale selection                              00
                                  00 = 250 dps
                                  01 = 500 dps
                                  10 = 2000 dps
                                  11 = 2000 dps
     0  SIM       SPI Mode (0=4-wire, 1=3-wire)                      0 */

  /* Adjust resolution if requested */
  switch(range)
  {
    case L3DS20_RANGE_250DPS:
      write8(L3GD20_REGISTER_CTRL_REG4, 0x00);
      break;
    case L3DS20_RANGE_500DPS:
      write8(L3GD20_REGISTER_CTRL_REG4, 0x10);
      break;
    case L3DS20_RANGE_2000DPS:
      write8(L3GD20_REGISTER_CTRL_REG4, 0x20);
      break;
  }
  /* ------------------------------------------------------------------ */
```

```
  /* Set CTRL_REG5 (0x24)
   ====================================================================
   BIT  Symbol    Description                                   Default
   ---  ------    --------------------------------------------- -------
     7  BOOT      Reboot memory content (0=normal, 1=reboot)        0
     6  FIFO_EN   FIFO enable (0=FIFO disable, 1=enable)            0
     4  HPen      High-pass filter enable (0=disable,1=enable)      0
   3-2  INT1_SEL  INT1 Selection config                            00
   1-0  OUT_SEL   Out selection config                             00 */

  /* Nothing to do ... keep default values */
  /* ------------------------------------------------------------------ */

  return true;
}

/***************************************************************************
 PUBLIC FUNCTIONS
 ***************************************************************************/
void Adafruit_L3GD20::read()
{
  uint8_t xhi, xlo, ylo, yhi, zlo, zhi;

  if (_cs == -1) {
    Wire.beginTransmission(address);
    // Make sure to set address auto-increment bit
    Wire.write(L3GD20_REGISTER_OUT_X_L | 0x80);
    Wire.endTransmission();
    Wire.requestFrom(address, (byte)6);

    // Wait around until enough data is available
    while (Wire.available() < 6);

    xlo = Wire.read();
    xhi = Wire.read();
    ylo = Wire.read();
    yhi = Wire.read();
    zlo = Wire.read();
    zhi = Wire.read();

  } else {
    digitalWrite(_clk, HIGH);
    digitalWrite(_cs, LOW);

    SPIxfer(L3GD20_REGISTER_OUT_X_L | 0x80 | 0x40); // SPI read, autoincrement
    delay(10);
    xlo = SPIxfer(0xFF);
    xhi = SPIxfer(0xFF);
    ylo = SPIxfer(0xFF);
    yhi = SPIxfer(0xFF);
    zlo = SPIxfer(0xFF);
    zhi = SPIxfer(0xFF);

    digitalWrite(_cs, HIGH);
  }
  // Shift values to create properly formed integer (low byte first)
  data.x = (xlo | (xhi << 8));
  data.y = (ylo | (yhi << 8));
  data.z = (zlo | (zhi << 8));

  // Compensate values depending on the resolution
  switch(range)
  {
    case L3DS20_RANGE_250DPS:
      data.x *= L3GD20_SENSITIVITY_250DPS;
      data.y *= L3GD20_SENSITIVITY_250DPS;
      data.z *= L3GD20_SENSITIVITY_250DPS;
      break;
    case L3DS20_RANGE_500DPS:
      data.x *= L3GD20_SENSITIVITY_500DPS;
      data.y *= L3GD20_SENSITIVITY_500DPS;
      data.z *= L3GD20_SENSITIVITY_500DPS;
      break;
    case L3DS20_RANGE_2000DPS:
      data.x *= L3GD20_SENSITIVITY_2000DPS;
      data.y *= L3GD20_SENSITIVITY_2000DPS;
      data.z *= L3GD20_SENSITIVITY_2000DPS;
      break;
  }
}

/***************************************************************************
 PRIVATE FUNCTIONS
 ***************************************************************************/
void Adafruit_L3GD20::write8(l3gd20Registers_t reg, byte value)
{
  if (_cs == -1) {
    // use i2c
    Wire.beginTransmission(address);
    Wire.write((byte)reg);
    Wire.write(value);
    Wire.endTransmission();
  } else {
    digitalWrite(_clk, HIGH);
    digitalWrite(_cs, LOW);

    SPIxfer(reg);
    SPIxfer(value);

    digitalWrite(_cs, HIGH);
  }
}

byte Adafruit_L3GD20::read8(l3gd20Registers_t reg)
{
  byte value;

  if (_cs == -1) {
    // use i2c
    Wire.beginTransmission(address);
    Wire.write((byte)reg);
    Wire.endTransmission();
    Wire.requestFrom(address, (byte)1);
    value = Wire.read();
    Wire.endTransmission();
  } else {
    digitalWrite(_clk, HIGH);
    digitalWrite(_cs, LOW);

    SPIxfer((uint8_t)reg | 0x80); // set READ bit
    value = SPIxfer(0xFF);

    digitalWrite(_cs, HIGH);
  }

  return value;
}
```

```
uint8_t Adafruit_L3GD20::SPIxfer(uint8_t x) {
  uint8_t value = 0;

  for (int i=7; i>=0; i--) {
    digitalWrite(_clk, LOW);
    if (x & (1<<i)) {
      digitalWrite(_mosi, HIGH);
    } else {
      digitalWrite(_mosi, LOW);
      }
    digitalWrite(_clk, HIGH);
    if (digitalRead(_miso))
      value |= (1<<i);
  }

  return value;
}
```

```
/***************************************************
  This is a library for the L3GD20 GYROSCOPE

  Designed specifically to work with the Adafruit L3GD20 Breakout
  ----> https://www.adafruit.com/products/1032

  These sensors use I2C or SPI to communicate, 2 pins (I2C)
  or 4 pins (SPI) are required to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Kevin "KTOWN" Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ****************************************************/
#ifndef __L3GD20_H__
#define __L3GD20_H__

#if (ARDUINO >= 100)
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif
#include "Wire.h"

#define L3GD20_ADDRESS                (0x6B)        // 1101011
#define L3GD20_POLL_TIMEOUT           (100)         // Maximum number of read
attempts
#define L3GD20_ID                     0xD4
#define L3GD20H_ID                    0xD7

#define L3GD20_SENSITIVITY_250DPS  (0.00875F)      // Roughly 22/256 for fixed
point match
#define L3GD20_SENSITIVITY_500DPS  (0.0175F)       // Roughly 45/256
#define L3GD20_SENSITIVITY_2000DPS (0.070F)        // Roughly 18/256
#define L3GD20_DPS_TO_RADS         (0.017453293F)  // degress/s to rad/s
multiplier

class Adafruit_L3GD20
{
  public:
    typedef enum
    {                                             // DEFAULT    TYPE
      L3GD20_REGISTER_WHO_AM_I            = 0x0F,   // 11010100   r
      L3GD20_REGISTER_CTRL_REG1           = 0x20,   // 00000111   rw
      L3GD20_REGISTER_CTRL_REG2           = 0x21,   // 00000000   rw
      L3GD20_REGISTER_CTRL_REG3           = 0x22,   // 00000000   rw
      L3GD20_REGISTER_CTRL_REG4           = 0x23,   // 00000000   rw
      L3GD20_REGISTER_CTRL_REG5           = 0x24,   // 00000000   rw
      L3GD20_REGISTER_REFERENCE           = 0x25,   // 00000000   rw
      L3GD20_REGISTER_OUT_TEMP            = 0x26,   //            r
      L3GD20_REGISTER_STATUS_REG          = 0x27,   //            r
      L3GD20_REGISTER_OUT_X_L             = 0x28,   //            r
      L3GD20_REGISTER_OUT_X_H             = 0x29,   //            r
      L3GD20_REGISTER_OUT_Y_L             = 0x2A,   //            r
      L3GD20_REGISTER_OUT_Y_H             = 0x2B,   //            r
      L3GD20_REGISTER_OUT_Z_L             = 0x2C,   //            r
      L3GD20_REGISTER_OUT_Z_H             = 0x2D,   //            r
      L3GD20_REGISTER_FIFO_CTRL_REG       = 0x2E,   // 00000000   rw
      L3GD20_REGISTER_FIFO_SRC_REG        = 0x2F,   //            r
      L3GD20_REGISTER_INT1_CFG            = 0x30,   // 00000000   rw
      L3GD20_REGISTER_INT1_SRC            = 0x31,   //            r
      L3GD20_REGISTER_TSH_XH              = 0x32,   // 00000000   rw
      L3GD20_REGISTER_TSH_XL              = 0x33,   // 00000000   rw
      L3GD20_REGISTER_TSH_YH              = 0x34,   // 00000000   rw
      L3GD20_REGISTER_TSH_YL              = 0x35,   // 00000000   rw
      L3GD20_REGISTER_TSH_ZH              = 0x36,   // 00000000   rw
      L3GD20_REGISTER_TSH_ZL              = 0x37,   // 00000000   rw
      L3GD20_REGISTER_INT1_DURATION       = 0x38    // 00000000   rw
    } l3gd20Registers_t;

    typedef enum
    {
      L3DS20_RANGE_250DPS,
      L3DS20_RANGE_500DPS,
      L3DS20_RANGE_2000DPS
    } l3gd20Range_t;

    typedef struct l3gd20Data_s
    {
      float x;
      float y;
      float z;
    } l3gd20Data;

    Adafruit_L3GD20(int8_t cs, int8_t mosi, int8_t miso, int8_t clk);
    Adafruit_L3GD20(void);

    bool begin(l3gd20Range_t rng=L3DS20_RANGE_250DPS, byte addr=L3GD20_ADDRESS);
    void read(void);

    l3gd20Data data;    // Last read will be available here

  private:
    void write8(l3gd20Registers_t reg, byte value);
    byte read8(l3gd20Registers_t reg);
    uint8_t SPIxfer(uint8_t x);
    byte address;
    l3gd20Range_t range;
    int8_t _miso, _mosi, _clk, _cs;
};

#endif
```

```
#include <Wire.h>
#include <Adafruit_L3GD20.h>

// define Arduino pins for SPI
#define GYRO_CS 4 // labeled CS - selects between SPI and I2C communication
#define GYRO_DO 5 // labeled SA0/SDO - serial data output
#define GYRO_DI 6 // labeled SDA/SDI - serial data input
#define GYRO_CLK 7 // labeled SCL - serial clock

#define GET_COORDS 8 // pin user3[5] on FPGA -- goes high if we are ready to
read coordinates

byte transmit_XL = 2;  // serial output user3[0] to FPGA for X[7:0]
byte transmit_XH = 3;  // serial output user3[1] to FPGA for X[15:8]
byte transmit_YL = 9;  // serial output user3[2] to FPGA for Y[7:0]
byte transmit_YH = 10; // serial output user3[3] to FPGA for Y[15:8]
byte transmit_ZL = 11; // serial output user3[6] to FPGA for Z[7:0]
byte transmit_ZH = 12; // serial output user3[7] to FPGA for Z[15:8]

byte mask = 1;  // our bit mask for serial transmission

// initialize gyroscope
Adafruit_L3GD20 gyro(GYRO_CS, GYRO_DO, GYRO_DI, GYRO_CLK);

// runs once
void setup() {
  Serial.begin(9600);
  pinMode(transmit_XL, OUTPUT);  // set up pin 2 on Arduino as output
  pinMode(transmit_XH, OUTPUT);  // set up pin 3 on Arduino as output
  pinMode(transmit_YL, OUTPUT);  // set up pin 9 on Arduino as output
  pinMode(transmit_YH, OUTPUT);  // set up pin 10 on Arduino as output
  pinMode(transmit_ZL, OUTPUT);  // set up pin 11 on Arduino as output
  pinMode(transmit_ZH, OUTPUT);  // set up pin 12 on Arduino as output
  pinMode(GET_COORDS, INPUT);  // set up pin 8 on Arduino as input

  // Try to initialize and send out an error if the chip was not detected (i.e.
incorrect wiring)
  if (!gyro.begin(gyro.L3DS20_RANGE_250DPS)) {
    Serial.println("Unable to initialize the L3GD20.  Check wiring!");
    pinMode(transmit_XL, OUTPUT);  // set up pin 2 on Arduino as output
    pinMode(transmit_XH, OUTPUT);  // set up pin 3 on Arduino as output
    pinMode(transmit_YL, OUTPUT);  // set up pin 9 on Arduino as output
    pinMode(transmit_YH, OUTPUT);  // set up pin 10 on Arduino as output
    pinMode(transmit_ZL, OUTPUT);  // set up pin 11 on Arduino as output
    pinMode(transmit_ZH, OUTPUT);  // set up pin 12 on Arduino as output
    while(1);
  }
}

// runs repeatedly
void loop() {
  gyro.read();

  int data_XL = (gyro.data.x) - 7;
  int data_XH = (data_XL >> 8);
  int data_YL = (gyro.data.y) + 6;
  int data_YH = (data_YL >> 8);
  int data_ZL = ((gyro.data.z) - 11);
  int data_ZH = (data_ZL >> 8);

  // transmit gyro coordinate readings if GET_COORDS is high
  if (digitalRead(GET_COORDS) == HIGH) {
    ////////////////////////////////////////////////////////////
    // Transmit XL data byte
```

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

Serial.print("data XL: ");
Serial.println(data_XL);
Serial.print("data XH: ");
Serial.println(data_XH);
Serial.print("data YL: ");
Serial.println(data_YL);
Serial.print("data YH: ");
Serial.println(data_YH);
Serial.print("data ZL: ");
Serial.println(data_ZL);
Serial.print("data ZH: ");
Serial.println(data_ZH);

// send start signal
digitalWrite(transmit_XL, HIGH);
delayMicroseconds(2400);
digitalWrite(transmit_XL, LOW);
delayMicroseconds(600);

// iterate through bit mask
for (mask = 00000001; mask > 0; mask <<= 1) {
  // if bitwise AND resolves to true
  if (data_XL & mask) {
    // send 1
    digitalWrite(transmit_XL, HIGH);
    delayMicroseconds(1200);
    digitalWrite(transmit_XL, LOW);
  }
  // if bitwise AND resolves to false
  else {
    // send 0
    digitalWrite(transmit_XL, HIGH);
    delayMicroseconds(600);
    digitalWrite(transmit_XL, LOW);
  }
  delayMicroseconds(600);
}

////////////////////////////////////////////////////////////
// Transmit XH data byte
////////////////////////////////////////////////////////////

// send start signal
digitalWrite(transmit_XH, HIGH);
delayMicroseconds(2400);
digitalWrite(transmit_XH, LOW);
delayMicroseconds(600);

// iterate through bit mask
for (mask = 00000001; mask > 0; mask <<= 1) {
  // if bitwise AND resolves to true
  if (data_XH & mask) {
    // send 1
    digitalWrite(transmit_XH, HIGH);
    delayMicroseconds(1200);
    digitalWrite(transmit_XH, LOW);
  }
  // if bitwise AND resolves to false
  else {
    // send 0
    digitalWrite(transmit_XH, HIGH);
    delayMicroseconds(600);
    digitalWrite(transmit_XH, LOW);
  }
  delayMicroseconds(600);
}

////////////////////////////////////////////////////////////
// Transmit YL data byte
////////////////////////////////////////////////////////////

// send start signal
digitalWrite(transmit_YL, HIGH);
delayMicroseconds(2400);
digitalWrite(transmit_YL, LOW);
delayMicroseconds(600);

// iterate through bit mask
for (mask = 00000001; mask > 0; mask <<= 1) {
  // if bitwise AND resolves to true
  if (data_YL & mask) {
    // send 1
    digitalWrite(transmit_YL, HIGH);
    delayMicroseconds(1200);
    digitalWrite(transmit_YL, LOW);
  }
  // if bitwise AND resolves to false
  else {
    // send 0
    digitalWrite(transmit_YL, HIGH);
    delayMicroseconds(600);
    digitalWrite(transmit_YL, LOW);
  }
  delayMicroseconds(600);
}

////////////////////////////////////////////////////////////////
// Transmit YH data byte
////////////////////////////////////////////////////////////////

// send start signal
digitalWrite(transmit_YH, HIGH);
delayMicroseconds(2400);
digitalWrite(transmit_YH, LOW);
delayMicroseconds(600);

// iterate through bit mask
for (mask = 00000001; mask > 0; mask <<= 1) {
  // if bitwise AND resolves to true
  if (data_YH & mask) {
    // send 1
    digitalWrite(transmit_YH, HIGH);
    delayMicroseconds(1200);
    digitalWrite(transmit_YH, LOW);
  }
  // if bitwise AND resolves to false
  else {
    // send 0
    digitalWrite(transmit_YH, HIGH);
    delayMicroseconds(600);
    digitalWrite(transmit_YH, LOW);
  }
  delayMicroseconds(600);
}

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

```
    // Transmit ZL data byte
    //////////////////////////////////////////////////////////////////

    // send start signal
    digitalWrite(transmit_ZL, HIGH);
    delayMicroseconds(2400);
    digitalWrite(transmit_ZL, LOW);
    delayMicroseconds(600);

    // iterate through bit mask
    for (mask = 00000001; mask > 0; mask <<= 1) {
      // if bitwise AND resolves to true
      if (data_ZL & mask) {
        // send 1
        digitalWrite(transmit_ZL, HIGH);
        delayMicroseconds(1200);
        digitalWrite(transmit_ZL, LOW);
      }
      // if bitwise AND resolves to false
      else {
        // send 0
        digitalWrite(transmit_ZL, HIGH);
        delayMicroseconds(600);
        digitalWrite(transmit_ZL, LOW);
      }
      delayMicroseconds(600);
    }

    ////////////////////////////////////////////////////////////////////
    // Transmit ZH data byte
    ////////////////////////////////////////////////////////////////////

    // send start signal
    digitalWrite(transmit_ZH, HIGH);
    delayMicroseconds(2400);
    digitalWrite(transmit_ZH, LOW);
    delayMicroseconds(600);

    // iterate through bit mask
    for (mask = 00000001; mask > 0; mask <<= 1) {
      // if bitwise AND resolves to true
      if (data_ZH & mask) {
        // send 1
        digitalWrite(transmit_ZH, HIGH);
        delayMicroseconds(1200);
        digitalWrite(transmit_ZH, LOW);
      }
      // if bitwise AND resolves to falase
      else {
        // send 0
        digitalWrite(transmit_ZH, HIGH);
        delayMicroseconds(600);
        digitalWrite(transmit_ZH, LOW);
      }
      delayMicroseconds(600);
    }
    delayMicroseconds(10000);
  }
}
```