Position Tracking using an Accelerometer

6.111 Final Report

Ertem Nusret Tas and Faysal Shair

December 2017

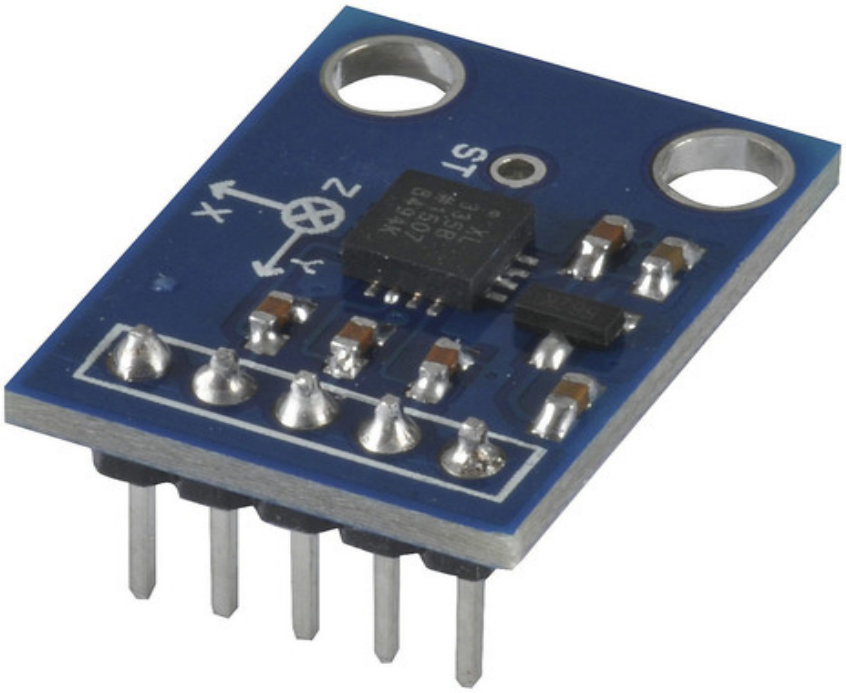
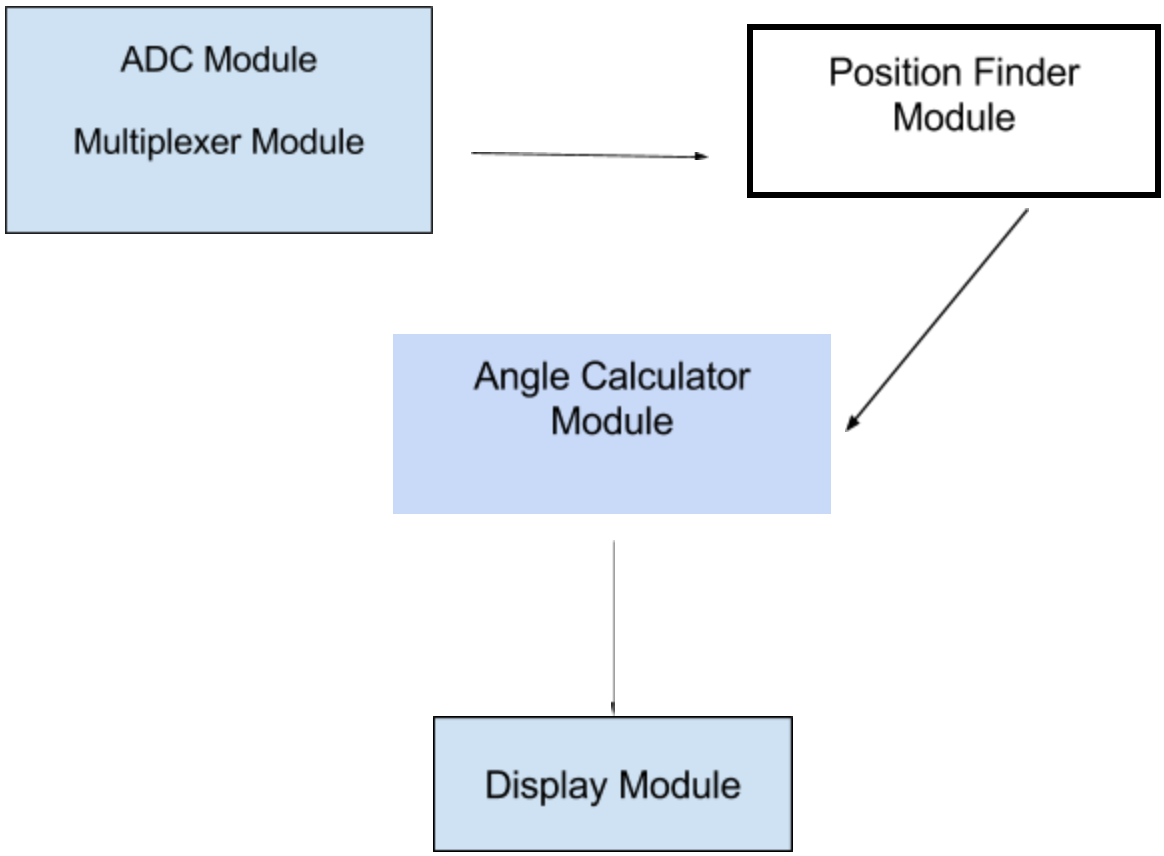


Table of Contents:

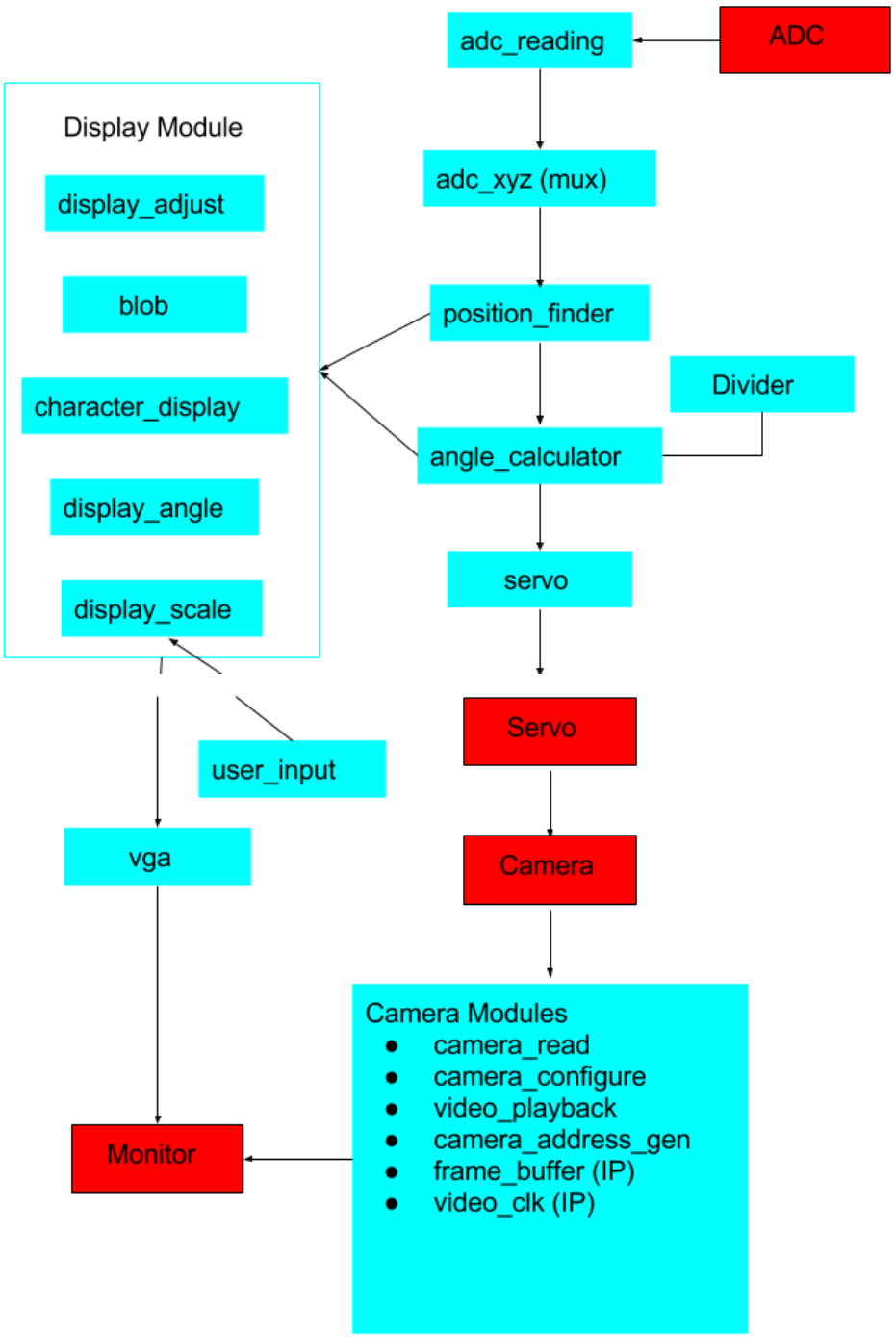
1) Overview

There are many different ways to monitor an object's position. Among those, GPS tracking is the first method that comes to mind, when we think of position tracking. But what if the GPS signal is momentarily lost? How would we predict the location of our moving object? We decided to address this issue by calculating an object's position locally using accelerometer data. This method of deriving position from acceleration and velocity is known, by electrical engineers, as a dead reckoning. Using accelerometer measurements to calculate position is a difficult task because the accelerometer is sensitive to both orientation and physical movements. As we were interested in observing physical movements rather than the orientation, an important part of our job was to process the raw data so that it could be used to correctly calculate the position. After calculating the x, y, and z coordinates, we computed the angle of the object in the x-y plane and the angle made by the ray between the object and the origin relative to the x-y plane.

Our design consisted of four main blocks of modules: the ADC/MUX modules, the position finder module, the angle calculator module, and the display modules. The schematic below gives a brief overview of the information flow through modules.



2) Block Diagram



In the block diagram, red boxes represent hardware units whereas blue boxes represent Verilog modules. Arrows represent the direction of information flow as the raw acceleration data is eventually processed to a visual display of location.

3) Module Implementations

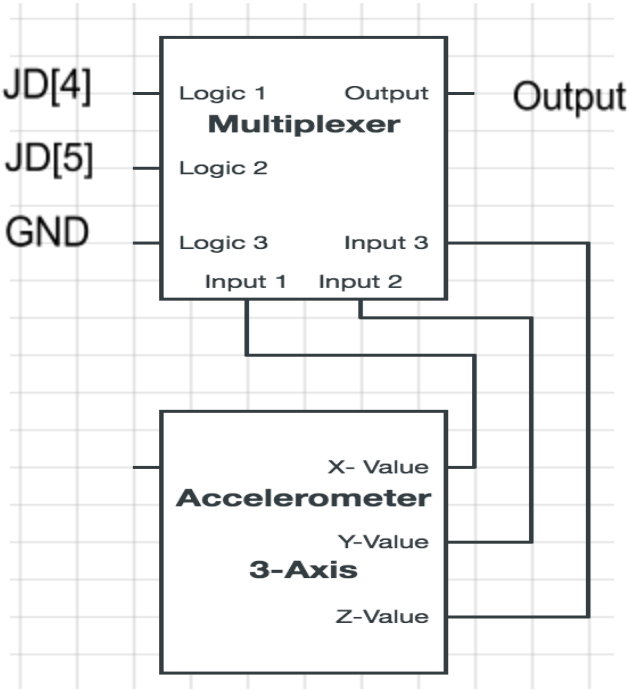
As mentioned in the Overview section, our position tracking system consists of four main blocks of modules. These are, consecutively, the ADC/MUX modules, the position finder module, the angle calculator module, and the display modules. The ADC/MUX module serves as an interface with the circuitry that produces the raw acceleration samples. The position finder module calculates the three dimensional position of the object whereas the angle calculator module finds the angles that are required to track the object from a fixed position. These two modules can be regarded as the "brain" of our position tracking system. Finally, display modules function as a graphical user interface and presents the data calculated by our system's brain in an intuitive way.

a) ADC and the Multiplexer Modules (adc_reading and adc_xyz) (Faysal)

Before explaining these modules, we would like to discuss the circuitry needed to transform the analog x, y and z signals from an external accelerometer into digital signals, which could be used by the Nexys 4 board as inputs. Four main chips were needed to get the job done: the analog-to-digital converter (ADC0804), the multiplexer (MAX14752), the 3-axis accelerometer, and the Op-amp (LM308).

Step 1: Multiplexer

Because the accelerometer has 3 outputs (x, y and z) we needed a multiplexer to select between the 3 analog outputs. The diagram below shows the accelerometer and multiplexer connection.

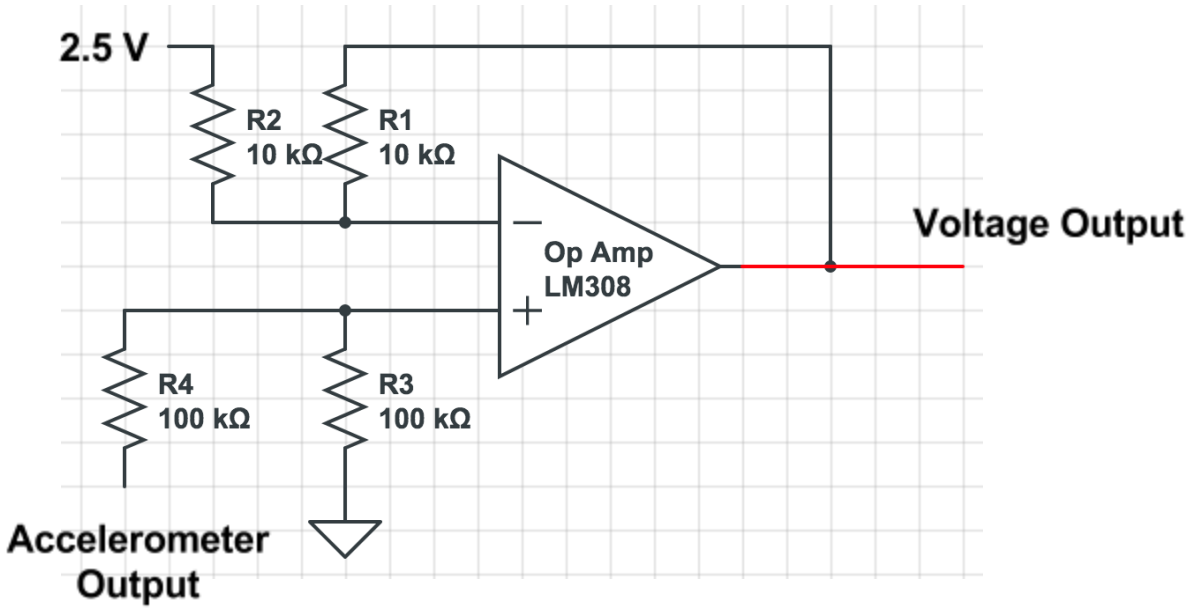


As we can see above, the accelerometer outputs its x, y and z values, and the multiplexer chooses which one to output based on the data at Logic 1, Logic 2, and Logic 3. We output the x-value when the binary at (logic 3, logic 2, logic 1) is (0,0,0), the y-value when the binary data is (0,0,1), and finally, the Z-value when the binary data is (0,1,0). Because of the constraints on the ADC, which will be discussed later, I sampled each output at 1 kHz (around 1000 samples of each x, y and z). I had Logic 3 grounded, while Logic 1 and Logic 2 were connected to JD[4] and JD[5] respectively. Th module that handles this basically waits for the ADC to finish computing its data. Once the ADC has computed the new value, we increase the binary count on JD[4] and JD[5], such that the multiplexer switches outputs. As I do that, I keep track of the values.

Step 2: Op-amp

The output of the accelerometer has weak voltage output swings, and a voltage offset, which makes the data messy. To deal with the offset which was at 5V, we decided to use a difference op-amp to subtract 2.5 V from the output. After reducing the offset from 5V to 2.5V, we amplified the signal, so that voltage swing could be more sensitive to movements. The circuit below shows these two processes

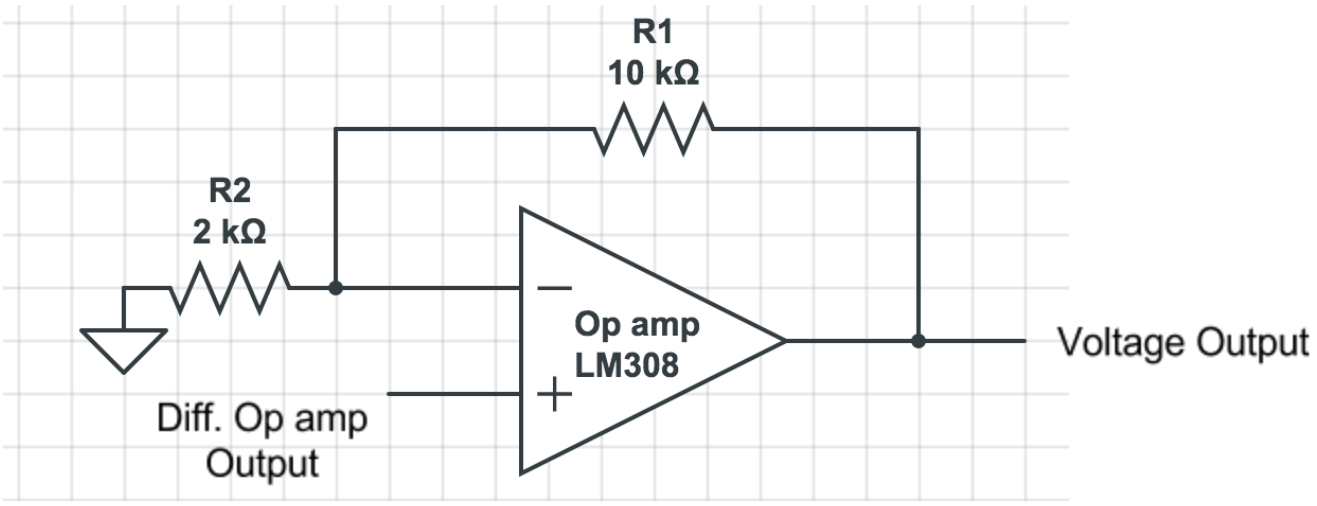
Differential Op-amp:



The circuit above is the differential Op Amp which is responsible for subtracting 2.5 V from the accelerometer outputs, therefore:

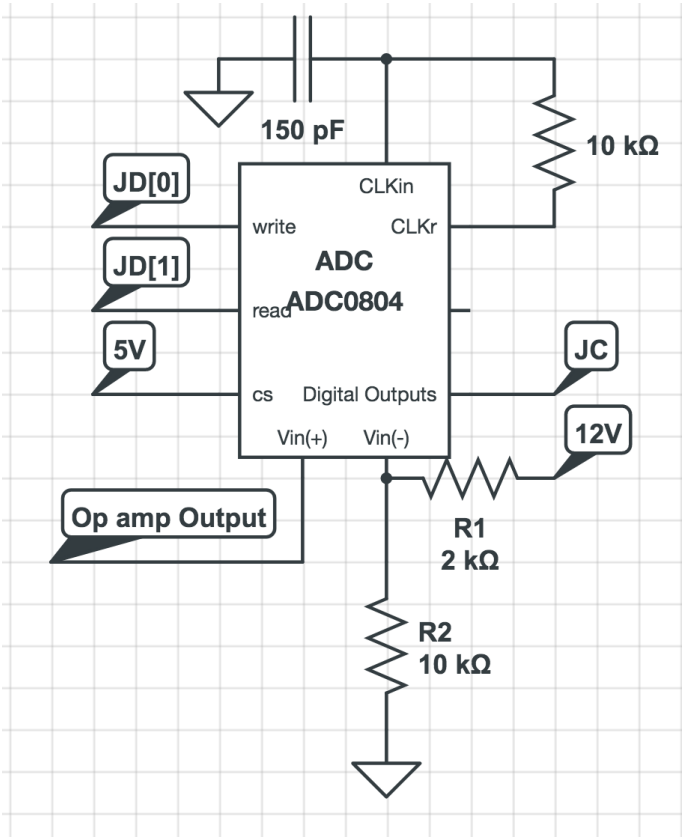
Voltage Output = Accelerometer Output - 2.5 V

Non-inverting Op Amp:



As we can see above, output of the differential amplifier is fed in as input to the non-inverting op amp. This signal is amplified by 1 + R1/R2 = 6. We now have a pronounced voltage swing, which we could now convert into a digital signal using the analog to digital converter. Although the signal now exceeds 5 V, this can be dealt with using the ADC, and will be discussed next.

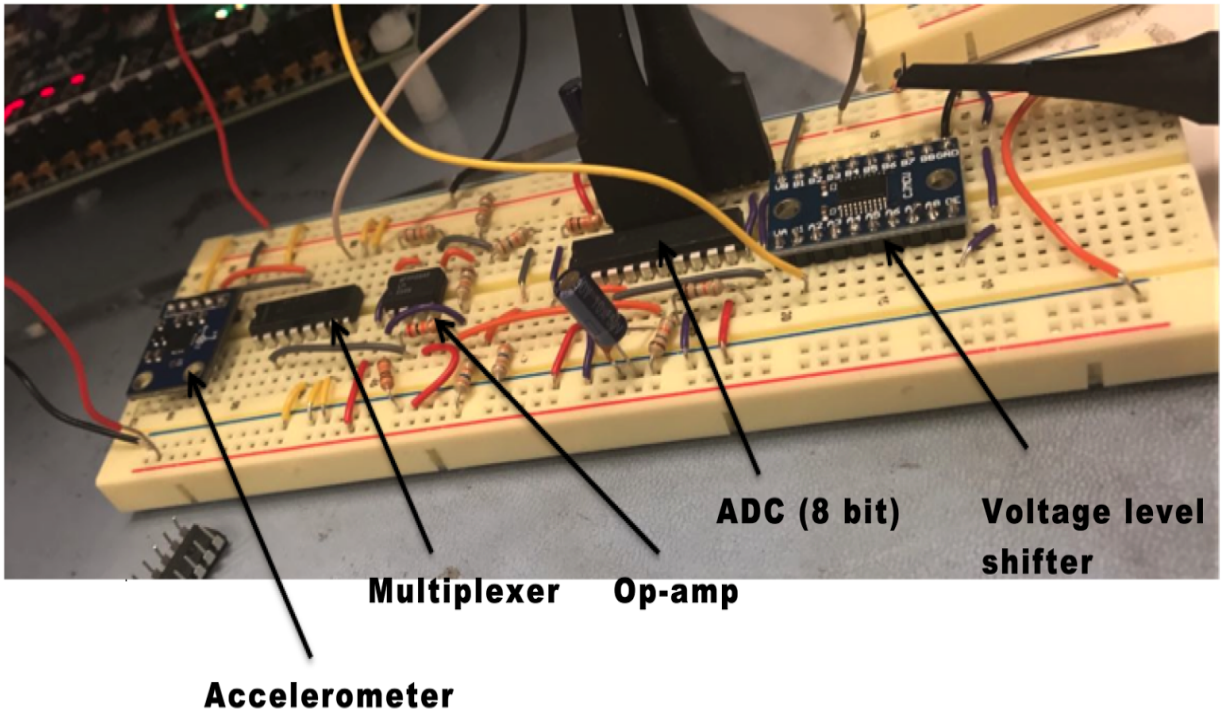
Analog-to-Digital Converter:



The ADC's relevant connection are shown above. I will now discuss some of these connections. The CLKr is the internal clock of the ADC, which I used as the input into the CLKin (the values I used for the capacitor and resistor were suggested by the datasheet). This ADC reads voltages between 0 and 5V, and has a resolution of 5/256. The input to the ADC is computed as Vin(+) - Vin(-), and since Vin(+) exceeds 5V, we had to keep a positive voltage at Vin(-). In this case, Vin(-) has a voltage of 10V. Of course the CS should be high. The write pin is connected to JD[0], while the read pin is connected to JD[1]. In order for the ADC to work, the write needs to transition from 0 to 1. Moreover, the write signal needs to be low and high for certain amounts of time. The read signal as well needs to transition from a high to a low in order to latch on the new values. Therefore in my code, I had the read and write signals change simultaneously, but in opposite direction (LOW to HIGH and High to LOW). I toggled them every 4095 cycles, which means that a new value was ready to be read by the ADC every 8190 cycles (3000 values per second). After asserting HIGH at the write pin and LOW at the read pin and waiting for 4095 cycles, a new value would be available, and latched onto the JC I/O of the Nexys 4 board (used as an input).

To know if the output is x, y or z coordinate, we created the multiplexer module, which basically looks at the the binary logic at the input to the MUX chip. The input to the MUX chip is simply JD[4] and JD[5]. After identifying if the signal is the x, y or z coordinate, I output one of these data coordinates.

The overall circuit is shown below:



b) Position Finder Module (position_finder) (Nusret)

The position finder module identifies the Cartesian coordinates of a tracked object by calculating the double integral of the acceleration samples. It accepts the object's initial position and 8 bit acceleration samples in x, y and z directions as input and outputs the object's velocity and position in real time. This module was specifically developed for the case where an object would be moved by a human hand from place to place within short distances.

As the value of the input acceleration samples changes between 0 and 255, the first step in the processing of this data is calibration. Calibration is achieved by subtracting the acceleration values corresponding to an immobile object from the arriving samples. These so-called calibration values are found by taking the average of the first 1024 samples while the object is at rest. Although they are automatically calculated after reset, we also implemented the functionality to recalculate them by pressing the BTNC (aka calibration) button. Once the differences between the incoming acceleration samples and the corresponding calibration values are found, results of this subtraction are passed onto the second step in which they are integrated into velocity and position values.

The simplest approach to the integration of acceleration samples is directly summing them to find the velocity in x, y and z axis. However, this approach has a number of problems associated with it. First, since this sum is calculated over discrete values, its result deviates from the true value of the integral due to the finite value of the sampling frequency. Although sampling rate can be increased to reduce the interval between consecutive acceleration samples, a higher sampling frequency is, also, more likely to produce noisy data. Hence, solution to the first problem results in a second problem, namely noise. However, avoiding noise comes at the cost of introducing non-existent drift velocities to our calculation due to the difference between the actual velocity and the Riemann sum of acceleration samples used to estimate it.

Although these problems can seriously mislead our calculation, they can still be alleviated by applying signal processing methods. Unfortunately, there exists another set of problems that are more subtle to identify and, thus, are harder to solve. Examples of these problems include incorrect measurements reported by the accelerometer or unexpected results due the way accelerometer was moved. For instance, if the accelerometer is only able to detect values that are above a certain threshold, then, an asymmetric movement where the object quickly speeds up and gradually slows down, would prompt an incorrect prediction for the object's position. Another example would be accelerometer having less sensitivity in certain directions due to hardware issues. Consequently, we need more than signal processing tools to build a functioning position finder module that can operate correctly under viable assumptions.

To cope with the issues identified above, five tools were used to process acceleration samples before integration: FIR filters, thresholds, gains, IIR filters and error terms. First, we realized that to filter out the noisy components of the acceleration signal, we would need a low pass filter. In this context, we implemented an FIR filter similar to the one used in Lab 5, where the coefficients were calculated by MATLAB's fir1 function. As in Lab 5, we stored the incoming samples in a buffer of size 31 and summed the scaled contents of this buffer to find the filter value corresponding to each sample. In this procedure, the cut-off frequency of our low pass filter was provided as an input to the fir1 function in the form of a Hamming window. For instance, if we desired to set the cutoff frequency at 95% percent of the Nyquist frequency of the samples, we would calculate the appropriate FIR coefficients by typing fir1(31, [0,0.95]) in MATLAB. (We multiplied these coefficients by 1024; in return, dividing the summed output by 1024.) Finally, after experimenting with various cutoff frequencies by using the fir1 function, we chose 80% of the Nyquist frequency as our cutoff as it yielded more stable results for velocity. However, considering the sampling frequency, it is possible to state that the low sampling rate for acceleration already removes most of the potential noise prior to the application of the filter.

Second, we implemented the functionality to remove samples magnitudes of which fall below a certain threshold. However, since we observed that the sensitivity of the accelerometer was already too low for motions we considered, the thresholds were lowered to zero. Nevertheless, with a more sensitive measurement device, this functionality might come in handy in the future.

Finally, we dealt with the problem of drift velocities via two channels. First, we added an error term to our summation as shown below:

$$v <= v + a_t + (a_t - a_{t-1}) / 2$$

where the velocity sample in any direction is updated by adding this error term besides the real time low-pass filtered acceleration data. Error terms are approximated by using the Trapezoidal rule for the value of acceleration between two samples. Despite reducing the drift velocity, they unfortunately failed to remove it completely. Consequently, we resorted to using a lossy integral for velocity which effectively corresponds to an IIR filter. However, as opposed to calculating the loss factor from the IIR filter coefficients, we simply experimented with different factors in the form $(2^n - 1) / 2^n$ where n changed from 7 to 13. Throughout this experiment, whereas high factors could not offset the extra drift introduced at each addition of an acceleration sample, low factors caused velocity to decay too fast to allow an accurate calculation of position. Consequently, we settled on the loss factor of 1023/1024 as our optimal value. Hence, the new formula for velocity became:

$$v <= ((2^n - 1) * v / 2^n) + a_t + (a_t - a_{t-1}) / 2$$

where n = 10.

Despite their success in removing noise and drift velocities, these solutions still offer many areas of improvement. For instance, it is possible to use more complex approximation methods such as Simpson's rule or non-linear regression while calculating the error term, which might then drive the drift velocity to zero. Second, a more rigorous approach can be adopted in the calculation of the loss factor. Due to our time constraints, we could not model the system holistically as a signals and system problem, instead having to rely on quick Matlab experiments as well as solutions found online about similar projects. Although we did run small simulations in Matlab to understand whether an undesirable low frequency / DC component was responsible for the drift velocity, these simulations used manually provided sample data that was adjusted to reflect those previously recorded on Verilog. Hence, as the data was limited in length, we eventually chose not to consider the simulation results while fine-tuning our FIR filter. In this context, a major enhancements could be analyzing the acceleration samples in real time with MATLAB over long time periods and designing filters informed by an accurate representation of acceleration in frequency domain.

Having found velocity samples from acceleration, we followed a very similar structure for going from velocity to position. However, here, we did not apply an IIR filter or decay factor as it is undesirable to have a decaying position value for the object. Furthermore, instead of using a low-pass filter, we utilized a bandpass filter with cutoff frequencies at 20% and 90% of the Nyquist frequencies as we wanted to;
i) get rid of the DC component of the velocity that corresponded to drift, and
ii) incorporate higher frequency components while excluding those that can be associated with noise.
Another difference compared to acceleration was that velocity samples were not scaled down by 1024 as we wanted to give display modules more freedom in choosing the sensitivity of position. Finally, again, no thresholds were applied here without any viable reason to use them. Incorporating these measures, final value of position can be expressed as below:

$$x <= x + v_t + (v_t - v_{t-1}) / 2$$

(It should be noted that the arguments suggested in the preceding paragraphs apply equally to acceleration, position and velocity values in x, y and z direction.)

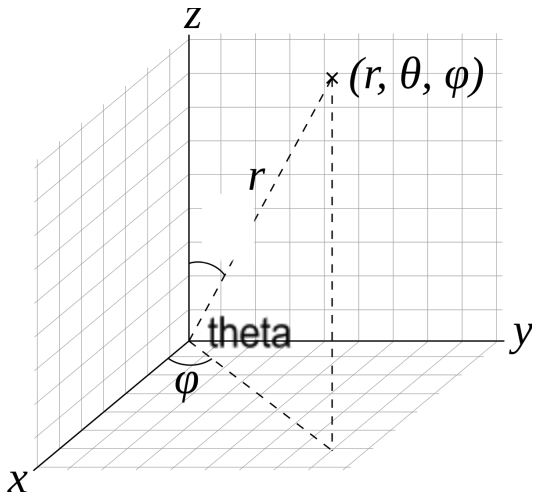
After the module was completed, our tests revealed an interesting phenomenon: the acceleration value sensed when the velocity and acceleration opposed each other seemed to be

larger than the case when they had the same sign. Since it is unlikely for an accelerometer to fail in this way, we assumed that this observation might be due to the movement of our hand that pushed the object. For instance, acceleration and deceleration might not have been symmetric or we could have been unintentionally tilting the device while moving it. (It is absolutely necessary to avoid tilting the accelerometer as it cannot detect the difference in acceleration coming from gravity or motion.) To overcome this problem, we added extra gain to the calibrated acceleration samples that have the same sign as velocity. Although we had to implement a hacky solution here due to our lack of information on the cause of this problem, it more or less fixed it after some tuning.

All in all, considering the low sensitivity of the accelerometer for human introduced motions, the module can be said to work well in all directions for our use case of abrupt and flat motions (less so in z). As we did not expect the object to move with constant velocity over long distances, we limited the functionality of our system to the objective described above. In the constant velocity case, we would have to remove the loss factor and most probably adjust our filters in a much different way. As mentioned in the first paragraph for position_finder, development of this module was very much informed by its use case as well as the nature of the accelerometer. (It had very low sensitivity unless it was tilted in which case our module worked with an even higher degree of accuracy.)

c) Angle Calculator Module (angle_calculator) (Faysal and Nusret)

Our ultimate goal here was to calculate two angles, theta and phi (shown in the figure below).



The phi angle is the angle between a point's projection on the x-y plane and the positive x-axis. This angle allows us to rotate the camera in the direction of the source. The theta angle is the

angle between the z-axis and the line connecting the point to the origin. This angle, on the other hand, allows us to tilt the camera such that it points to the height of the point.

There are several ways to find these angles using the x,y and z coordinates, but we did this using the trigonometric functions (tangent function). To compute the value of the tangent function, we used a lookup table, which contained a list of tangent values for angles between 0 to 85. Because the tangent of angles greater than 85 degrees is very big (infinite for tan(90)), we approximated angles above 85 degrees as being 90 degrees. Moreover because the tangent values contain ratios, all these values were multiplied by 1024.

Since our coordinate values are signed values, the first thing done by the angle calculator module is to check if the values are positive or negative. If the value is negative, it takes its complement and adds 1, casting it as an unsigned value. If the value is positive it simply takes the unsigned version of the coordinate.

Calculating Phi:

For this, we basically compute the error = ((y -coordinate *1024) - tan(angle) * x-coordinate), and try to minimize it by iterating through angles. We start with angle = 0, then increment the angle by 1 after every clock cycle, and use the look-up table to find tan(angle). If the error term is smaller than the previous minimum value, then we save it as our new minimum value. After going through all 85 angles in the lookup table, the value of angle that minimizes this error term becomes our phi angle. Now we have an angle in the first quadrant (between 0 and 90 degrees) that needs to be adjusted over 360 degrees to represent the sign of the coordinate. The table below shows the final changes we make to the value after finding the signs of x and y:

x-coordinate	y-coordinate	Modifications to Angle
Positive	Positive	Angle
Positive	Negative	360 - Angle
Negative	Positive	180 - Angle
Negative	Negative	180 + angle

After making these modifications, we have an angle that ranges from 0 to 360 degrees. Because this can take up to around 90 clk cycles, we assert done1, when the angle is ready to be taken by the servo.

Calculation of Theta:

The discussion of theta is very similar to phi. Again, we compute the error term, which is now error = ((z-coordinate * 1024) - d*tan(angle)). In this case, d is the distance of the object's x-y projection from the origin in the x-y plane. We compute d by taking the square root of the squared sum of x and y. Again at each clock cycle we calculate the error using the lookup table and increment the angle by 1 at the end of the step. After going through all 85 values, the angle that minimizes the error value will be our phi value. Again, this value is between 0 and 90 degrees, but needs to be adjusted over 180 degrees. Once again, the table shows the modification we had to make to the value after accounting for the sign of z.

z-coordinate	Modification to Angle
Positive	90 + angle
Negative	90 - angle

Now the angle can vary between 0 and 180 degrees giving us a the ability to move the servo such that it faces the object's height.

Because it can take up to 90 clock cycles to compute the theta angle, we have a done2 output which is asserted once the angle is ready.

In order to prevent constant angle fluctuation, we hold the value of the angle for half a second, then recompute the new angle again. This ensure that any spikes in the coordinates will not be translated into angles. To do this we added a divider clock module which simply outputs a square wave with a frequency of 1 Hz as a ready signal for angle calculation.

d) Display Modules (Faysal and Nusret)

Display modules collectively enable the display of the object's position, x and y axis, their scales, characters for object's position and angles and, finally, the camera image. They are combined in a header module named display module. It contains initializations of the following modules which are described below in detail:

- display_adjust, (along with blob, used to display object location)
- blob, (used to display the axis and the object location)
- display_scale, (displays an adjustable scale on the axis)

- display_angle, (displays a line from the origin to the blob representing the object)
- character_display (displays characters on the screen) and,
- the modules used for camera image

i) Displaying the Axis (display_adjust, blob) (Faysal)

To create the x-axis and y-axis, we simply used the blob module which was provided in Lab 3. We had to make a few modifications to the module to make it compatible with the Nexys 4 labkit. After creating the x-axis and y-axis, we moved on to displaying the object in the x-y plane where it was represented as a square blob with its x and y positions assigned by the display_adjust module.

Because the x and y coordinates are signed values, but the screen pixel count consists only of positive values, we had to rearrange our data based on the sign.

The first step was to choose maximum values for the x and y coordinates, such that when these max values are attained, the blob would be on the bottom right of the screen.

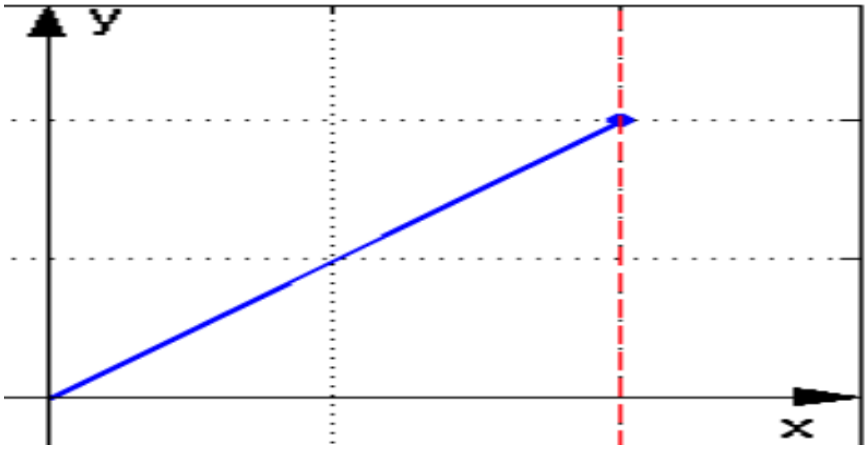
Therefore: x-position = ((x_coordinate)/(max x)) * width/2 (signed value)
Y-position = ((y_coordinate)/(max y)) * length/2 (signed value)

If the x-position is positive, then we simply add the x-coordinate to the width/2. We do this because the x-position on the screen increases from 0 (left) to 640 (right). If the x-coordinate is negative, then we subtract the x-coordinate from the width/2. In both cases, we check that the x-position of the blob did not exceed 640 or go below 0. This ensured that the blob stayed visible at all times. If the blob's x-position goes beyond 640, then we cap its x-position at 640, and if its x-position goes below 0, then again we keep it capped at 0.

If the y-position is positive, then we subtract its y-position from length/2 and make sure the y-position is not less than 0. If the y-position is negative, then we add its absolute value to length/2, and we make sure the y-position does not exceed 480. Therefore we keep the y-position capped at 0 (if y-position < 0) or 480 (if y-position > 480).

ii) Displaying the Angle Indicator (display_angle) (Faysal)

This is simply the line connecting the object to the origin, making it easier to see the angle made between the object and the x-axis. This was difficult to implement because it required the use of ratios. Instead of using division, we chose to multiply two sides and take their difference.



To make this more clear, to draw this point we need the (hcount - width) to (vcount - length) ratio to match the ratio of (x_display - width) to (y_display - length) . In other words, we need (hcount - width)/(vcount- length) = (x_display - width)/(y_display - length). But because dealing with ratios in verilog is difficult, we decided to compute;
(hcount-width) * (y_display-length) = (vcount-length) * (x_display-width). In order to create line thickness we need a slight difference between the two sides of the equation.
Therefore: Error = |(hcount-width) * (y_display-length) =(vcount-length) * (x_display-width)|. By verifying that the set of hcount and vcount values are within that error, we assign the pixel to our specified color. Moreover we need to check that the hcount and vcount values are indeed between the object and origin.

iii) Displaying the Axis Scales (display_scale) (Nusret)

The display_scale module takes hcount, vcount and interval values as its input and outputs the pixel values corresponding to the hcount and vcount values at the given clock cycle. These pixels are assigned color so as to display scales on the x and y axis with the interval distance specified by the interval input.

Although it is fairly trivial to display scales at constant intervals by calling the blob module multiple times, parametrizing this interval requires real time update of the position of the next blob while the VGA scans through the screen from up to bottom. Hence, this module first calculates the numbers of scales that has to be placed on each axis and initializes a count register to hold this number. This register's value is used to find the location of the next scale while the monitor screen is scanned. Hence, the counter register for the scales on the y axis

decreases with each pass of the vcount through a scale. The same procedure is applied to the scales on the x axis as well with each pass of the hcount. Once the counter reaches zero, it starts to increase to display the scales on the negative x and y axis. The same procedure is repeated after the counter reaches back to its initial value.

Although the module works well, it entails a system reset to update its intervals when the interval is changed during system operation. This is due to the fact that number of scales to be placed on each axis is calculated at reset. Hence, when the interval is changed mid process, the number of scales with the new interval distances equal the old number of scales, leaving portions of the axis blank if the interval was decreased. Although this is not an important problem as a scale shift would most likely be accompanied by a system reset, it is desirable to correct this in future iterations of the system as it is an easily solvable problem. A logic gate that checks the provided interval values and recalculates the new scale number upon a change in this value would be more than sufficient to fix the problem.

iv) Displaying Characters (character_display) (Nusret)

The character_display module creates a numeric display of the tracked object's x, y and z coordinates, as well as the angles theta and phi described in the section for the Angle Calculator Module. It consists of two main parts.

In the first part, the binary numbers corresponding to the angle and position values are converted into decimal numbers as decimals are easier to interpret by the people. In this context, angles are represented as three digit numbers since the maximum angle value is 360 and coordinates are displayed as four digit numbers since the position inputs consist of 12 signed bits changing between -2048 and 2047. Since the number of digits for each value is known, it is possible to calculate the value of each digit in the decimal representation by subtracting that digit's value from the original binary values. For instance, while calculating the decimal numbers for the coordinates, the module first subtracts 1000 from the input binary number until it is shrinked below 1000. Then, it subtracts 100 and so on until all the values corresponding to the coordinate digits are stored in the dedicated registers.

Once the decimal representations are generated and stored for the aforementioned values, the module calls the char_string_display module taken from the class website to display these representations on the screen. Since char_string_display requires the ascii values of the characters it would display, character display module refers to a lookup table to find the ascii value corresponding to the digits in the decimal representations of position and angle. Finally, after the positions of the characters are specified in the char_string_display module,

character_display puts the characters for the x, y and z coordinates and the angles as positive or negative values on the screen.

One issue with the display is that since char_string_display was originally designed to display characters in a video setting, it introduces glitches in the VGA display. Although it was proposed by the course staff that inputting an hcount value delayed by 2 or 3 clock cycles to char_string_display might solve this problem, we still continued to observe vertical glitches on the screen after implementing this suggestion. However, as this was a minor issue, we did not put more effort into solving this problem. Furthermore, it is also possible that the solution might require an in-depth understanding of the char_string-display module which involves complex features.

v) Servo Module (Faysal)

The servo responds to changes in a pulse modulated width waveform. A PWM waveform is simply a square wave characterized by a duty cycle. The duty cycle is the portion of the time the wave is high over the period of the wave. In my design, the square wave always has a period of 0.00350s. When the signal is high for 1 ms, and low for the remaining period (2.5ms), the servo displays a 0 degree angle. When the signal is high for 1.5ms and low for the remaining period the servo displays a angle of 90 degrees and finally when the signal is high for 2 ms, the angle is 180 degrees.

That said, the servo basically feeds off the phi angle calculated by the angle calculator module, and uses this value to calculate its duty cycle, and thus display the square wave. In the servo module, I first wait for done1 to be asserted (which means that the angle has been computed by the angle calculator). I then use this formula to calculate the high portion of the graph
high portion = (angle/9)*1250 + 25000, I then calculate the low portion which is simply period - high portion = 87500 - low portion. As we can see, regardless of the value of angle, after adding high portion to low portion we always end up with 87500 clock cycles, which is the time for the waveform period. After calculating the high and low portion times, I set the PWM as 1 during the high portion time, then 0 during the low portion time.

Unfortunately, we did not have the servo tilt (angle of theta) because the z-coordinate value, which theta depends on, was very sensitive to all kinds of motions and tilts. That said, we chose to leave out the tilt (up/down) and only accounted for the rotation (left/right).

vi) Camera Modules (Faysal and Nusret)

The modules required for displaying the camera image via Nexys 4 board were provided by Weston Braun. However, we had to merge his top module with our top module that was copied over from Lab 4 and modified according to our needs. Moreover, we had to create IP blocks for BRAM and system clock as the IPs in Weston's original project could not be transferred directly to our project. Finally, we had to wire the camera to the JA and JB ports of the Nexys 4 board and attach it to the servo which, henceforth, contributed to the display functionality of the project.

4) Lessons Learned and Advice for Future Projects

After working with the accelerometer, we came to two realizations. First, we realized that the accelerometer is more sensitive to tilt (gravity) than physical movements. Second, we acknowledged the difficulty of modeling constant velocity with an accelerometer.

There is actually not much that can be done to filter out the effects of tilt by only using an accelerometer. However, there exists a device, a gyroscope, that only detects tilt and not physical movements. By coupling the gyroscope with the accelerometer, we can account for the effects of tilt. This can be done by adjusting the voltage on the gyroscope such that its voltage output reacts exactly the same way the accelerometer does to tilt. By subtracting the gyroscope output voltage from the accelerometer output voltage, we can finally isolate physical movements from tilt.

The reason why the accelerometer was poor at detecting constant velocity even without a lossy integral, is because there is no such thing as constant velocity in the real world. In our design, we expected to see four changes in acceleration for every movement: a sudden increase in acceleration, acceleration dropping to zero (constant velocity), deceleration and deceleration rising to zero (stopping). In this context, we predicted to see zero acceleration at constant velocity. The problem, however, was that the surface we moved the accelerometer across was never totally smooth, and our hand motions were far from steady. These two limitations made the accelerometer constantly see positive and negative acceleration, while our code expected to see zero acceleration. This made it hard for the accelerometer to truly represent constant velocity. Instead, the accelerometer responded very well to sudden and short-lasting changes in motion (quick pushes).

When it comes to solving this issue a few things come to mind. The first way to address this is to put a threshold on the acceleration. This means that acceleration values below this threshold will be overlooked. While we did have a threshold value for acceleration, we could not use it because the acceleration values were not as sensitive as we wanted them to be. A lot of the acceleration resolution was lost by the 8-bit ADC. In the future, we would probably use a 16-bit ADC, which is more sensitive to acceleration values.

Finally, instead of moving the accelerometer with our hands on the lab desk, we can, in the future, build a track and have the ADC moved by a motor. This would hopefully make the ADC move more smoothly, allowing us to see motion closer to "constant velocity".

5) Summary

In short, we were able to use the accelerometer to track sudden changes in motion, and use this information to predict the motion of the moving object. We also had the object's position in the x-y plane displayed on the monitor, with numerical values for its x-position and y-position coordinates. Finally, we hooked up a servo to track the object in 2D space, and had a camera mounted on the servo to follow the object. This feature could be added to any moving object, that needs to be followed in 2D space.

We really enjoyed working on this project, which has greatly improved our lab, FPGA, and problem solving skills.

We would also like to thank Gim Hom, Madeleine Waller, Joe Steinmeyer, Weston Braun (especially for the camera code) and the rest of the TA's and LA's for their guidance and patience throughout the project.

6) References:

http://www.chrobotics.com/library/accel-position-velocity

https://pdfs.semanticscholar.org/e0dc/4b1b973f65da612382f6b07e441cd30e339b.pdf

Vivado IP Catalog

APPENDIX: VERILOG CODE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Project: Position Finder System
// Updated 9/29/2017 V2.0
// Create Date: 10/1/2015 V1.0
// Design Name:
// Module Name: labkit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module labkit(
  input CLK100MHZ,
  input[15:0] SW,
  input BTNC, BTNU, BTNL, BTNR, BTND,
  output[3:0] VGA_R,
  output[3:0] VGA_B,
  output[3:0] VGA_G,
  output[7:0] JD,
  input [7:0] JC,
  inout [7:0] JA,
  inout [7:0] JB,
  output VGA_HS,
  output VGA_VS,
  output LED16_B, LED16_G, LED16_R,
  output LED17_B, LED17_G, LED17_R,
  output[15:0] LED,
  output[7:0] SEG,  // segments A-G (0-6), DP (7)
  output[7:0] AN    // Display 0-7
  );
```

```
// create 25mhz system clock
   wire clock_25mhz;
   clock_quarter_divider clockgen(.clk100_mhz(CLK100MHZ), .clock_25mhz(clock_25mhz));

//  instantiate 7-segment display;
   wire [31:0] data;
   wire [6:0] segments;
   display_8hex display(.clk(clock_25mhz),.data(data), .seg(segments), .strobe(AN));
   assign SEG[6:0] = segments;
   assign SEG[7] = 1'b1;


//////////////////////////////////////////////////////////////////////////////////
//
//  PROJECT: POSITION FINDER SYSTEM
//
//////////////////////////////////////////////////////////////////////////////////

parameter xhighbit = 31;
parameter xlowbit = 21;
parameter yhighbit = 31;
parameter ylowbit = 21;

////////////////////////////wire definitions/////////////////////////////////////

   wire reset;
   wire calibrate;

   wire [7:0] accx;
   wire [7:0] accy;
   wire [7:0] accz;

   wire signed [64:0] x_0 = 0;
   wire signed [64:0] y_0 = 0;
   wire signed [64:0] z_0 = 0;

   wire signed [32:0] vx, vy, vz;
   wire signed [64:0] x, y, z;
   wire signed [11:0] dx;
   wire signed [11:0] dy;
   wire signed [11:0] dz;
```

```
    wire [10:0] degree1, degree2;
    wire calc_angle;
    wire degree_done_1, degree_done_2;

    wire scale1, scale2, scale3;
    wire [7:0] interval;
    wire [9:0] hcount;
    wire [9:0] vcount;
    wire hsync, vsync, at_display_area;
    wire [3:0] G_R;
    wire [3:0] G_G;
    wire [3:0] G_B;

    wire btnu;
    wire btnd;
    wire camera;

debounce #(.DELAY(250000)) db5(.reset(1'b0),.clock(clock_25mhz),.noisy(SW[15]),.clean(reset));
debounce #(.DELAY(250000)) db6(.reset(reset),.clock(clock_25mhz),.noisy(SW[0]),.clean(camera));
debounce #(.DELAY(250000)) dbx(.reset(reset),.clock(clock_25mhz),.noisy(SW[1]),.clean(scale1));
debounce #(.DELAY(250000)) dby(.reset(reset),.clock(clock_25mhz),.noisy(SW[2]),.clean(scale2));
debounce #(.DELAY(250000)) dbz(.reset(reset),.clock(clock_25mhz),.noisy(SW[3]),.clean(scale3));
debounce #(.DELAY(250000)) db7(.reset(reset),.clock(clock_25mhz),.noisy(BTNC),.clean(calibrate));
debounce #(.DELAY(250000)) db8(.reset(reset),.clock(clock_25mhz),.noisy(BTNU),.clean(btnu));
debounce #(.DELAY(250000)) db9(.reset(reset),.clock(clock_25mhz),.noisy(BTND),.clean(btnd));

assign dx = $signed({x[64],x[xhighbit:xlowbit]});
assign dy = $signed({y[64],y[yhighbit:ylowbit]});
assign dz = $signed({z[64],z[yhighbit:ylowbit]});

//////////////////////////module instantiations for getting acceleration
data//////////////////////////////////////////////

wire read;
wire write;
wire ready;
wire [1:0] mux;
wire [1:0] xyz_out;
wire check2;

adc_reading adc_inst(.clk(clock_25mhz), .write(JD[0]), .read(read), .mux(JD[5:4]), .xyz_mux(xyz_out),
.ready(ready), .check2(JD[6]));
```

```
wire [7:0] coordinate;
wire [7:0] x_coor;
wire [7:0] y_coor;
wire [7:0] z_coor;
wire check;

adc_xyz adcxyz(.clk(clock_25mhz), .ready(ready), .xyz_mux(xyz_out), .coordinate(JC), .x_coor(accx), .y_coor(accy), .z_coor(accz), .check(JD[7]));

//////////////////////////taking user input for scale//////////////////////////////////////////////////////////////////////

user_input ui(.clock(clock_25mhz), .scale({scale3, scale2, scale1}), .interval(interval));

//////////////////////////position_finder instantiation for calculating position//////////////////////////////////////////////

position_finder #(.N(8), .M(32)) pf1(.clock(clock_25mhz), .calibrate(calibrate), .reset(reset), .data_in(ready),
            .accx_in(accx), .accy_in(accy), .accz_in(accz), .x_0(x_0), .y_0(y_0), .z_0(z_0), .x(x), .y(y), .z(z),
            .vx(vx), .vy(vy), .vz(vz), .led_debug1(LED[0]), .led_debug2(LED[1]));

//////////////////////////angle_calculator instantiation for calculating angle/////////////////////////////////////////////////

Divider #(.max(5000000)) d1(.clock(clock_25mhz), .reset(reset), .div_clock(calc_angle));
angle_calculator #(.M(11)) ac1(.x_in(dx), .y_in(dy), .z_in(dz), .clock(clock_25mhz), .calc_clock(calc_angle),
                                .degree1(degree1), .degree2(degree2), .done1(degree_done_1), .done2(degree_done_2));

//////////////////////////display instantiation for displaying graph and its constituents//////////////////////////////////////

display #(.ylowbit(ylowbit), .yhighbit(yhighbit), .xlowbit(xlowbit), .xhighbit(xhighbit)) disp(.clock(clock_25mhz), .reset(reset),
                .hcount(hcount), .vcount(vcount), .interval(interval), .x(x), .y(y), .z(z), .G_R(G_R), .G_G(G_G), .G_B(G_B));

//////////////////////////code for generating and getting the camera image////////////////////////////////////////////////////
```

```
//camera display signals
wire [3:0] pix_CB;
wire [3:0] pix_CG;
wire [3:0] pix_CR;
wire c_hsync;
wire c_vsync;

//camera signals
wire video_clock;
wire camera_pwdn;
wire camera_clk_in;
wire camera_clk_out;
wire [7:0] camera_dout;
wire camera_scl, camera_sda;
wire camera_vsync, camera_hsync;
wire [15:0] camera_pixel;
wire camera_pixel_valid;
wire camera_reset;
wire camera_frame_done;

assign camera_clk_in = video_clock;
assign camera_pwdn = 0;
assign camera_reset = ~reset;

//assign camera outputs
assign JA[0] = camera_pwdn;
assign camera_dout[0] = JA[1];
assign camera_dout[2] = JA[2];
assign camera_dout[4] = JA[3];
assign JA[4] = camera_reset;
assign camera_dout[1] = JA[5];
assign camera_dout[3] = JA[6];
assign camera_dout[5] = JA[7];

assign camera_dout[6] = JB[0];
assign JB[1] = camera_clk_in;
assign camera_hsync = JB[2];
//assign JB[3]= camera_sda; //unused
assign camera_dout[7] = JB[4];
assign camera_clk_out = JB[7];
assign camera_vsync = JB[5];
//assign JB[7] = camera_scl; //unused
```

```
wire [11:0] memory_read_data;
wire [11:0] memory_write_data;
wire [18:0] memory_read_addr;
wire [18:0] memory_write_addr;
wire memory_write_enable;

//clock generation
 video_clk video_clk_1 (
   .clk_in1(CLK100MHZ),
   .clk_out1(video_clock)
   );

//camera configuration module
camera_configure camera_configure_1 (
   .clk(video_clock),
   .start(btnd),
   .sioc(JB[6]),
   .siod(JB[3]),
   .done()
   );

//camera interface
camera_read camera_read_1 (
   .p_clock(camera_clk_out),
   .vsync(camera_vsync),
   .href(camera_hsync),
   .p_data(camera_dout),
   .pixel_data(camera_pixel),
   .pixel_valid(camera_pixel_valid),
   .frame_done(camera_frame_done)
   );

//write camera data to frame buffer
camera_address_gen camera_address_gen_1 (
   .camera_clk(camera_clk_out),
   .camera_pixel_valid(camera_pixel_valid),
   .camera_frame_done(camera_frame_done),
   .capture_frame(~(btnu)),
   .camera_pixel(camera_pixel),
   .memory_data(memory_write_data),
   .memory_addr(memory_write_addr),
   .memory_we(memory_write_enable)
```

```
   );

video_playback video_playback_1 (
   .pixel_data(memory_read_data),
   .video_clk(video_clock),
   .memory_addr(memory_read_addr),
   .vsync(c_vsync),
   .hsync(c_hsync),
   .video_out({pix_CR, pix_CG, pix_CB})
   );


//frame buffer memory
frame_buffer frame_buffer_1 (
   .clka(camera_clk_out),
   .wea(memory_write_enable),
   .addra(memory_write_addr),
   .dina(memory_write_data),
   .clkb(video_clock),
   .enb(1'b1),
   .addrb(memory_read_addr),
   .doutb(memory_read_data)
   );


//////////////////////////angle calculator instantiation for
servo//////////////////////////////////////////////////

wire [10:0] servo_degree1;
wire [10:0] servo_degree2;
wire done11;
wire done22;

angle_calculator #(.M(11))
        angle_inst (.x_in(dx), .y_in(dy) , .z_in(dz), .clock(clock_25mhz),
                .calc_clock(calc_angle), .degree1(servo_degree1),
                 .degree2(servo_degree2), .done1(done11), .done2(done22));

//////////////////////////////////servo_instance////////////////////////////////////////////////////////////
//////////

servo serv_inst (.clk(clock_25mhz), .angle(servo_degree1), .pwm(JD[3]));
```

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

   vga vga1(.vga_clock(clock_25mhz),.hcount(hcount),.vcount(vcount),
        .hsync(hsync),.vsync(vsync),.at_display_area(at_display_area));

   assign VGA_HS = camera ? c_hsync : ~hsync;
   assign VGA_VS = camera ? c_vsync : ~vsync;

   assign VGA_R = camera ? pix_CR : G_R;
   assign VGA_G = camera ? pix_CG : G_G;
   assign VGA_B = camera ? pix_CB : G_B;

endmodule

//////////////////////////////////////////////////////////////////////////////
//
//user_input module for taking user's choice on scaling of the graph
//
//////////////////////////////////////////////////////////////////////////////

module user_input (
   input clock,
   input [2:0] scale,
   output reg [7:0] interval);

   always @(posedge clock) begin
      case(scale)
         3'b000: interval <= 8'd10;
         3'b001: interval <= 8'd15;
         3'b010: interval <= 8'd20;
         3'b011: interval <= 8'd25;
         3'b100: interval <= 8'd30;
         3'b101: interval <= 8'd35;
         3'b110: interval <= 8'd40;
         3'b111: interval <= 8'd50;
      endcase
   end
endmodule

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

```
//display module for generating the graph and its constituents
//
//////////////////////////////////////////////////////////////////////////////////

module display  #(parameter ylowbit = 21, yhighbit = 31, xlowbit = 21, xhighbit = 31)(
   input clock,
   input reset,
   input [9:0] hcount,
   input [9:0] vcount,
   input [7:0] interval,
   input signed [64:0] x,
   input signed [64:0] y,
   input signed [64:0] z,
   output [3:0] G_R,
   output [3:0] G_G,
   output [3:0] G_B
   );

   //wires for pixel values
   wire [3:0] VGA_R1;
   wire [3:0] VGA_B1;
   wire [3:0] VGA_G1;

   wire [3:0] VGA_R2;
   wire [3:0] VGA_B2;
   wire [3:0] VGA_G2;

   wire [3:0] VGA_R3;
   wire [3:0] VGA_B3;
   wire [3:0] VGA_G3;

   wire [3:0] VGA_G4;
   wire [3:0] VGA_B4;
   wire [3:0] VGA_R4;

   wire [3:0] VGA_G5;
   wire [3:0] VGA_B5;
   wire [3:0] VGA_R5;

   wire [3:0] pix_B;
   wire [3:0] pix_G;
   wire [3:0] pix_R;
```

```
   wire [3:0] pix_C;
   wire [2:0] pix_char;

   wire [9:0] x_display;
   wire [9:0] y_display;

   wire signed [11:0] dx;
   wire signed [11:0] dy;
   wire signed [11:0] dz;

   //registers
   reg [9:0] hcount1;
   reg [9:0] hcount2;
   reg [9:0] hcount3;
   reg [9:0] hcount4;

   assign pix_C = {1'b1, pix_char};
   assign dx = $signed({x[64],x[xhighbit:xlowbit]});
   assign dy = $signed({y[64],y[yhighbit:ylowbit]});
   assign dz = $signed({z[64],z[yhighbit:ylowbit]});

   //Buffer on hcount to prevent glitch appearence of characters on the screen
   always @(posedge clock) begin
      if(reset) begin
         hcount1 <= 0;
         hcount2 <= 0;
         hcount3 <= 0;
         hcount4 <= 0;
      end
      else begin
         hcount1 <= hcount;
         hcount2 <= hcount1;
         hcount3 <= hcount2;
         hcount4 <= hcount3;
      end
   end

   //module instantiation for scaling object location to graph locations
         display_adjust #(.ylowbit(ylowbit), .yhighbit(yhighbit), .xlowbit(xlowbit), .xhighbit(xhighbit))
dis_instance(.clock(clock),
                                 .reset(reset), .x(x), .y(y), .x_display(x_display), .y_display(y_display));
```

```
  //blob for y axis
                                                blob            #(.WIDTH(5),.HEIGHT(523),.COLOR(12'hF_F_F))
wiki(.x_blob(10'd322),.y_blob(10'd0),.hcount(hcount),.vcount(vcount),
                                        .pix_R(VGA_R1), .pix_G(VGA_G1), .pix_B(VGA_B1), .clk(clock));

  //blob for x axis
                                                blob            #(.WIDTH(750),.HEIGHT(5),.COLOR(12'hF_F_F))
wiki2(.x_blob(10'd0),.y_blob(10'd238),.hcount(hcount),.vcount(vcount),
                                        .pix_R(VGA_R2), .pix_G(VGA_G2), .pix_B(VGA_B2), .clk(clock));

  //blob for the rectangle representing the object at the graph locations found
                                                blob            #(.WIDTH(20),.HEIGHT(20),.COLOR(12'hF_F_F))
wiki3(.x_blob(x_display),.y_blob(y_display),.hcount(hcount),.vcount(vcount),
                                        .pix_R(VGA_R3), .pix_G(VGA_G3), .pix_B(VGA_B3), .clk(clock));

  //module instantiation for displaying scale
      display_scale ds1(.clk(clock), .reset(reset), .hcount(hcount), .vcount(vcount), .interval(interval),
.pixel_R(VGA_R5),
            .pixel_G(VGA_G5), .pixel_B(VGA_B5));

  //module instantiation for displaying the line from origin to the object location
          display_angle dis_ang_inst(.clk(clock), .reset(reset), .hcount(hcount), .vcount(vcount),
.x_display(x_display),
                .y_display(y_display),.pixel_R(VGA_R4), .pixel_G(VGA_G4), .pixel_B(VGA_B4));

  //module for displaying characters on the screen
    character_display cd1(.reset(reset), .clock(clock), .hcount(hcount4), .vcount(vcount), .x(dx), .y(dy),
.z(dz),
              .angle1(degree1), .angle2(degree2), .pixel(pix_char));

  assign G_R = (VGA_R1 | VGA_R2 | VGA_R3 | VGA_R4 | pix_C | VGA_R5);
  assign G_G = (VGA_G1 | VGA_G2 | VGA_G3 | VGA_G4 | pix_C | VGA_G5);
  assign G_B = (VGA_B1 | VGA_B2 | VGA_B3 | VGA_B4 | pix_C | VGA_B5);

endmodule

//////////////////////////////////////////////////////////////////////////////
//
//clock_quarter_divider module (taken from lab4)
//
//////////////////////////////////////////////////////////////////////////////

module clock_quarter_divider(input clk100_mhz, output reg clock_25mhz = 0);
```

```
   reg counter = 0;

   // VERY BAD VERILOG
   // VERY BAD VERILOG
   // VERY BAD VERILOG
   // But it's a quick and dirty way to create a 25Mhz clock
   // Please use the IP Clock Wizard under FPGA Features/Clocking
   //
   // For 1 Hz pulse, it's okay to use a counter to create the pulse as in
   // assign onehz = (counter == 100_000_000);
   // be sure to have the right number of bits.

   always @(posedge clk100_mhz) begin
      counter <= counter + 1;
      if (counter == 0) begin
         clock_25mhz <= ~clock_25mhz;
      end
   end
endmodule

//////////////////////////////////////////////////////////////////////////////
//
//vga module (taken from lab4)
//
//////////////////////////////////////////////////////////////////////////////

module vga(input vga_clock,
          output reg [9:0] hcount = 0,    // pixel number on current line
          output reg [9:0] vcount = 0,    // line number
          output reg vsync, hsync,
          output at_display_area);

  // Comments applies to XVGA 1024x768, left in for reference
  // horizontal: 1344 pixels total
  // display 1024 pixels per line
  reg hblank,vblank;
  wire hsyncon,hsyncoff,hreset,hblankon;
  assign hblankon = (hcount == 639);    // active H  1023
  assign hsyncon = (hcount == 655);     // active H + FP 1047
  assign hsyncoff = (hcount == 751);    // active H + fp + sync  1183
  assign hreset = (hcount == 799);      // active H + fp + sync + bp 1343

  // vertical: 806 lines total
```

```
  // display 768 lines
  wire vsyncon,vsyncoff,vreset,vblankon;
  assign vblankon = hreset & (vcount == 479);    // active V   767
  assign vsyncon = hreset & (vcount ==490 );     // active V + fp   776
  assign vsyncoff = hreset & (vcount == 492);    // active V + fp + sync  783
  assign vreset = hreset & (vcount == 523);      // active V + fp + sync + bp 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 vga_clock) 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

  end

  assign at_display_area = ((hcount >= 0) && (hcount < 640) && (vcount >= 0) && (vcount < 480));

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
//Divider module produces a high signal periodically given the period in terms of
//clock cycles. It was taken from lab4 and used to adjust the period of update for
//angles.
//
//////////////////////////////////////////////////////////////////////////////////

module Divider #(parameter max = 25000000)
   (input clock,
          input reset,
          output reg div_clock);

          reg [24:0] count;

          always @(posedge clock) begin
                  if(reset) begin
```

```
                div_clock <= 0;
                count <= 0;
        end
        else begin
                if(count == max) begin
                        count <= 0;
                        div_clock <= 1;
                end
                else begin
                        count <= count + 1;
                        div_clock <= 0;
                end
        end
    end

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
//position_finder module calculates the double integral of the acceleration data
//to find the position of the object
//
//Its precision and response to different types of acceleration data is controlled
//by parameters. These parameters are:
//
//accfilter and vfilter: filter sizes for the FIR filters used to process acceleration
//and velocity data. They are set to 31 for the FIR-31 filters with coefficients coming
//from look-up modules
//
//accthresholdp, accthresholdn, vthresholdp, vthresholdn: thresholds for considering acceleration
//and velocity data above and below zero during integration. They are set 0 as the acceleration
//sensitivity to slow human motion is already low (emprically tested).
//
//vdecayn and vdecaym: This is an indirect way to implement an IIR filter. The ratio vdecayn/vdecaym
//determines how fast the velocity decays when there is no acceleration data. This method enhances the
//systems precision for fast, discreet movements of the objects (like a person moving things with his
hand).

//gainh, gainl, gainm: gainh/gainm and gainl/gainm are the high and low gains that are used to offset
//inherent asymmetries in the raw acceleration data. For instance, we have consistently observed that
direction
//of acceleration and velocity affect the sensitivity of the accelerometer. To prevent any distortion of
results,
```

```
//these gains were used to bring sensitivities to the same level. They can also be used to increase or decrease the
//overall sensitivities of our system.
//
//////////////////////////////////////////////////////////////////////////////////

module position_finder #(parameter N = 8, M = 32)
   (input clock,
   input calibrate,
   input reset,
   input data_in,
   input signed [N-1:0] accx_in,
   input signed [N-1:0] accy_in,
   input signed [N-1:0] accz_in,
   input signed [64:0] x_0,
   input signed [64:0] y_0,
   input signed [64:0] z_0, //be aware of gravity
   output reg signed [64:0] x,
   output reg signed [64:0] y,
   output reg signed [64:0] z,
   output reg signed [M:0] vx,
   output reg signed [M:0] vy,
   output reg signed [M:0] vz,
   output reg led_debug1,
   output reg led_debug2
   );

   //Parameters
   parameter duration = 1024;
   parameter vdecayn = 1023;
   parameter vdecaym = 1024;


   parameter accfilter = 31; //original 31
         parameter ascaling = 1024; //original 1024
   parameter accthresholdp = 0;
   parameter accthresholdn = 0;

   parameter vfilter = 31;
         parameter vscaling = 1;
   parameter vthresholdp = 0;
   parameter vthresholdn = 0;
```

```
        parameter gainh = 2;
        parameter gainl = 1;
        parameter gainm = 2;

        parameter L = M;
        parameter B = L;

//Registers
reg signed [L:0] vxd;
reg signed [L:0] vyd;
reg signed [L:0] vzd;

reg signed [B:0] accx;
reg signed [B:0] accy;
reg signed [B:0] accz;

reg signed [64:0] waitx;
reg signed [64:0] waity;
reg signed [64:0] waitz;

reg signed [64:0] waitvx;
reg signed [64:0] waitvy;
reg signed [64:0] waitvz;

reg signed [64:0] vtempx;
reg signed [64:0] vtempy;
reg signed [64:0] vtempz;

reg signed [64:0] vxa;
reg signed [64:0] vya;
reg signed [64:0] vza;

reg signed [64:0] vxb;
reg signed [64:0] vyb;
reg signed [64:0] vzb;

reg [vfilter*(L+1)-1:0] vxlist;
reg [vfilter*(L+1)-1:0] vylist;
reg [vfilter*(L+1)-1:0] vzlist;

reg signed [64:0] sumvx;
reg signed [64:0] sumvy;
reg signed [64:0] sumvz;
```

```
reg signed [64:0] atempx;
reg signed [64:0] atempy;
reg signed [64:0] atempz;

reg signed [64:0] accxa;
reg signed [64:0] accya;
reg signed [64:0] accza;

reg signed [64:0] accxb;
reg signed [64:0] accyb;
reg signed [64:0] acczb;

reg signed [B:0] accx0;
reg signed [B:0] accy0;
reg signed [B:0] accz0;

reg signed [64:0] taccx0;
reg signed [64:0] taccy0;
reg signed [64:0] taccz0;

reg [accfilter*(B+1)-1:0] accxlist;
reg [accfilter*(B+1)-1:0] accylist;
reg [accfilter*(B+1)-1:0] acczlist;

reg signed [64:0] sumaccx;
reg signed [64:0] sumaccy;
reg signed [64:0] sumaccz;

reg ready;
reg [31:0] count;

        reg init;
        reg signal;

        reg asumsignal;
        reg [5:0] acounter = 6'd0;
        reg vsumsignal;
reg [5:0] vcounter = 6'd0;

wire signed [10:0] coeffa;
wire [5:0] indexa;
assign indexa = acounter;
```

```
wire signed [16:0] coeffv;
wire [5:0] indexv;
assign indexv = vcounter;

//FIR filters
coeffsa31 cfa31(.index(indexa),.coeff(coeffa));
coeffsv31 cfv31(.index(indexv),.coeff(coeffv));

always @ (posedge clock) begin

   accx <= $signed({1'b0,accx_in});
   accy <= $signed({1'b0,accy_in});
   accz <= $signed({1'b0,accz_in});

   if(reset) begin
                              init <= 0;
      led_debug1 <= 0;
      led_debug2 <= 0;
      signal <= 0;
      ready <= 0;
      count <= 0;

      acounter <= 0;
      asumsignal <= 0;
      vcounter <= 0;
      vsumsignal <= 0;

      waitx <= 0;
      waity <= 0;
      waitz <= 0;

      waitvx <= 0;
      waitvy <= 0;
      waitvz <= 0;

      accxa <= 0;
      accya <= 0;
      accza <= 0;

      accxb <= 0;
      accyb <= 0;
      acczb <= 0;
```

```
sumaccx <= 0;
sumaccy <= 0;
sumaccz <= 0;

accxlist <= 0;
accylist <= 0;
acczlist <= 0;

x <= x_0;
y <= y_0;
z <= z_0;

accx0 <= 0;
accy0 <= 0;
accz0 <= 0;

taccx0 <= 0;
taccy0 <= 0;
taccz0 <= 0;

vx <= 0;
vy <= 0;
vz <= 0;

vxa <= 0;
vya <= 0;
vza <= 0;

vxb <= 0;
vyb <= 0;
vzb <= 0;

sumvx <= 0;
sumvy <= 0;
sumvz <= 0;

vxlist <= 0;
vylist <= 0;
vzlist <= 0;

atempx <= 0;
atempy <= 0;
```

```
    atempz <= 0;

    vtempx <= 0;
    vtempy <= 0;
    vtempz <= 0;
  end
  else if(calibrate | ~ready) begin
    led_debug1 <= 0;
    led_debug2 <= 1;
    if(calibrate) begin
      count <= 0;
      ready <= 0;
      taccx0 <= 0;
      taccy0 <= 0;
      taccz0 <= 0;
                              init <= 0;
    end
    else if(count == duration) begin
      ready <= 1;
                              init <= 1;
      accx0 <= taccx0 / duration;
      accy0 <= taccy0 / duration;
      accz0 <= taccz0 / duration;
    end
    else if(data_in) begin
      count <= count + 1;
      taccx0 <= taccx0 + accx;
      taccy0 <= taccy0 + accy;
      taccz0 <= taccz0 + accz;
    end
  end
  else if(ready) begin

    led_debug1 <= 1;
    led_debug2 <= 1;

                            if(init) begin
      init <= 0;
      atempx <= 0;
      atempy <= 0;
      atempz <= 0;
      vtempx <= 0;
      vtempy <= 0;
```

```
            vtempz <= 0;
            accxb <= 0;
            accyb <= 0;
            acczb <= 0;
            accxlist <= 0;
            accylist <= 0;
            acczlist <= 0;
        end
                                else if(data_in) begin
            signal <= 1;
            vx <= $signed({waitvx[64],waitvx[M-1:0]});
            vy <= $signed({waitvy[64],waitvy[M-1:0]});
            vz <= $signed({waitvz[64],waitvz[M-1:0]});

            vxd <= $signed({waitvx[64],waitvx[L-1:0]});
            vyd <= $signed({waitvy[64],waitvy[L-1:0]});
            vzd <= $signed({waitvz[64],waitvz[L-1:0]});

            acounter <= 0;

            if(accfilter > 1) begin
               asumsignal <= 1; //implment positive/negative displacement difference
                  if((accx - accx0 > 0 && accx - accx0 <= accthresholdp) || (accx - accx0 < 0 && accx - accx0 >=
-1*accthresholdn))  accxlist  <=  {accxlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accx0  -
accx0)/gainm)};
                        else if((vxd >= 0 && accx - accx0 >= 0) || (vxd <= 0 && accx - accx0 <= 0)) accxlist <=
{accxlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accx - accx0)/gainm)};
                        else if((vxd >= 0 && accx - accx0 <= 0) || (vxd <= 0 && accx - accx0 >= 0)) accxlist <=
{accxlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainl*(accx - accx0)/gainm)};

                  if((accy - accy0 > 0 && accy - accy0 <= accthresholdp) || (accy - accy0 < 0 && accy - accy0 >=
-1*accthresholdn))  accylist  <=  {accylist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accy0  -
accy0)/gainm)};
                        else if((vyd >= 0 && accy - accy0 >= 0) || (vyd <= 0 && accy - accy0 <= 0)) accylist <=
{accylist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accy - accy0)/gainm)};
                        else if((vyd >= 0 && accy - accy0 <= 0) || (vyd <= 0 && accy - accy0 >= 0)) accylist <=
{accylist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainl*(accy - accy0)/gainm)};

                   if((accz - accz0 > 0 && accz - accz0 <= accthresholdp) || (accz - accz0 < 0 && accz - accz0 >=
-accthresholdn)) acczlist <= {acczlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accz0 - accz0)/gainm)};
                        else if((vzd >= 0 && accz - accz0 >= 0) || (vzd <= 0 && accz - accz0 <= 0))  acczlist <=
{acczlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainh*(accz - accz0)/gainm)};
                        else if((vzd >= 0 && accz - accz0 <= 0) || (vzd <= 0 && accz - accz0 >= 0))  acczlist <=
```

```
{acczlist[accfilter*(B+1)-(B+1)-1:0],$unsigned(gainl*(accz - accz0)/gainm)};

            end
            else begin
                 if((accx - accx0 > 0 && accx - accx0 <= accthresholdp) || (accx - accx0 < 0 && accx - accx0 >=
-1*accthresholdn)) accxa <= 0;
               else if((vxd >= 0 && accx - accx0 >= 0) || (vxd <= 0 && accx - accx0 <= 0)) accxa <= gainh*(accx
- accx0)/gainm;
                else if((vxd >= 0 && accx - accx0 <= 0) || (vxd <= 0 && accx - accx0 >= 0)) accxa <= gainl*(accx
- accx0)/gainm;

                 if((accy - accy0 > 0 && accy - accy0 <= accthresholdp) || (accy - accy0 < 0 && accy - accy0 >=
-1*accthresholdn)) accya <= 0;
                      else if((vyd >= 0 && accy - accy0 >= 0) || (vyd <= 0 && accy - accy0 <= 0)) accya <=
gainh*(accy - accy0)/gainm;
                else if((vyd >= 0 && accy - accy0 <= 0) || (vyd <= 0 && accy - accy0 >= 0)) accya <= gainl*(accy
- accy0)/gainm;

                  if((accz - accz0 > 0 && accz - accz0 <= accthresholdp) || (accz - accz0 < 0 && accz - accz0 >=
-1*accthresholdn)) accza <= 0;
                else if((vzd >= 0 && accz - accz0 >= 0) || (vzd <= 0 && accz - accz0 <= 0)) accza <= gainh*(accz
- accz0)/gainm;
               else if((vzd >= 0 && accz - accz0 <= 0) || (vzd <= 0 && accz - accz0 >= 0)) accza <= gainl*(accz -
accz0)/gainm;
            end

            sumaccx <= 0;
            sumaccy <= 0;
            sumaccz <= 0;
            accxb <= accxa;
            accyb <= accya;
            acczb <= accza;
          end

          else if(asumsignal) begin
            if(acounter > accfilter - 1) begin
               asumsignal <= 0;
                                      accxa <= sumaccx / ascaling;
               accya <= sumaccy / ascaling;
               accza <= sumaccz / ascaling;
            end
            else begin
               sumaccx <= sumaccx + (coeffa * $signed(accxlist[(acounter*(B+1))+:B+1]));
```

```
            sumaccy <= sumaccy + (coeffa * $signed(accylist[(acounter*(B+1))+:B+1]));
            sumaccz <= sumaccz + (coeffa * $signed(acczlist[(acounter*(B+1))+:B+1]));
            acounter <= acounter + 1;
         end
      end

      waitvx <= ((vx * vdecayn) / vdecaym) + accxa + ((accxa - accxb) / 2);
      waitvy <= ((vy * vdecayn) / vdecaym) + accya + ((accya - accyb) / 2);
      waitvz <= ((vz * vdecayn) / vdecaym) + accza + ((accza - acczb) / 2);

      if(signal) begin
         signal <= 0;
         x <= waitx;
         y <= waity;
         z <= waitz;
         vcounter <= 0;
         if(vfilter > 1) begin
            vsumsignal <= 1;
                  if((vxd > 0 && vxd <= vthresholdp) || (vxd < 0 && vxd >= -1*vthresholdn)) vxlist <=
{vxlist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vxd-vxd)};
            else vxlist <= {vxlist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vxd)};
                  if((vyd > 0 && vyd <= vthresholdp) || (vyd < 0 && vyd >= -1*vthresholdn)) vylist <=
{vylist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vyd-vyd)};
            else vylist <= {vylist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vyd)};
                  if((vzd > 0 && vzd <= vthresholdp) || (vzd < 0 && vzd >= -1*vthresholdn)) vzlist <=
{vzlist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vzd-vzd)};
            else vzlist <= {vzlist[vfilter*(L+1)-(L+1)-1:0],$unsigned(vzd)};
         end
         else begin
            if((vxd > 0 && vxd <= vthresholdp) || (vxd < 0 && vxd >= -1*vthresholdn)) vxa <= 0;
            else vxa <= vxd;
            if((vyd > 0 && vyd <= vthresholdp) || (vyd < 0 && vyd >= -1*vthresholdn)) vya <= 0;
            else vya <= vyd;
            if((vzd > 0 && vzd <= vthresholdp) || (vzd < 0 && vzd >= -1*vthresholdn)) vza <= 0;
            else vza <= vzd;
         end
         sumvx <= 0;
         sumvy <= 0;
         sumvz <= 0;
         vxb <= vxa;
         vyb <= vya;
         vzb <= vza;
      end
```

```
        else if(vsumsignal) begin
          if(vcounter > vfilter - 1) begin
            vsumsignal <= 0;
            vxa <= -1 * sumvx / vscaling;
            vya <= -1 * sumvy / vscaling;
            vza <= -1 * sumvz / vscaling;
          end
          else begin
            sumvx <= sumvx + (coeffv * $signed(vxlist[vcounter*(L+1)+:L+1]));
            sumvy <= sumvy + (coeffv * $signed(vylist[vcounter*(L+1)+:L+1]));
            sumvz <= sumvz + (coeffv * $signed(vzlist[vcounter*(L+1)+:L+1]));
            vcounter <= vcounter + 1;
          end
        end

        waitx <= x + vxa + ((vxa - vxb) / 2);
        waity <= y + vya + ((vya - vyb) / 2);
        waitz <= z + vza + ((vza - vzb) / 2);
      end
    end
endmodule


//////////////////////////////////////////////////////////////////////////////////
//
//coeffsa31 gives the coefficients of the FIR-31 filter used while calculating
//velocity from acceleration
//
//////////////////////////////////////////////////////////////////////////////////

module coeffsa31(
  input wire [5:0] index,
  output reg signed [10:0] coeff
   );
  always @(index) begin
    case (index)

      //Low pass filter at 0.8 of Nyquist frequency
      6'd0:  coeff = 11'sd0;
      6'd1:  coeff = -11'sd1;
      6'd2:  coeff = 11'sd3;
      6'd3:  coeff = -11'sd4;
      6'd4:  coeff = 11'sd4;
```

```
    6'd5:  coeff = 11'sd0;
    6'd6:  coeff = -11'sd8;
    6'd7:  coeff = 11'sd19;
    6'd8:  coeff = -11'sd26;
    6'd9:  coeff = 11'sd22;
    6'd10: coeff = 11'sd0;
    6'd11: coeff = -11'sd41;
    6'd12: coeff = 11'sd94;
    6'd13: coeff = -11'sd149;
    6'd14: coeff = 11'sd190;
    6'd15: coeff = 11'sd820;
    6'd16: coeff = 11'sd190;
    6'd17: coeff = -11'sd149;
    6'd18: coeff = 11'sd94;
    6'd19: coeff = -11'sd41;
    6'd20: coeff = 11'sd0;
    6'd21: coeff = 11'sd22;
    6'd22: coeff = -11'sd26;
    6'd23: coeff = 11'sd19;
    6'd24: coeff = -11'sd8;
    6'd25: coeff = 11'sd0;
    6'd26: coeff = 11'sd4;
    6'd27: coeff = -11'sd4;
    6'd28: coeff = 11'sd3;
    6'd29: coeff = -11'sd1;
    6'd30: coeff = 11'sd0;
    default: coeff = 11'hXXX;
   endcase
  end
endmodule


//////////////////////////////////////////////////////////////////////////////
//
//coeffsv31 gives the coefficients of the FIR-31 filter used while calculating
//position from velocity
//
//////////////////////////////////////////////////////////////////////////////

module coeffsv31(
  input wire [5:0] index,
  output reg signed [16:0] coeff
   );
```

```
always @(index) begin
  case (index)

    //Band pass filter at [0.2,0.9] of Nyquist frequency
    6'd0:  coeff = -17'sd111;
    6'd1:  coeff = 17'sd49;
    6'd2:  coeff = -17'sd338;
    6'd3:  coeff = -17'sd106;
    6'd4:  coeff = -17'sd395;
    6'd5:  coeff = 17'sd0;
    6'd6:  coeff = 17'sd827;
    6'd7:  coeff = 17'sd466;
    6'd8:  coeff = 17'sd3083;
    6'd9:  coeff = -17'sd861;
    6'd10: coeff = 17'sd3211;
    6'd11: coeff = -17'sd6800;
    6'd12: coeff = -17'sd900;
    6'd13: coeff = -17'sd15403;
    6'd14: coeff = -17'sd5753;
    6'd15: coeff = 17'sd45848;
    6'd16: coeff = -17'sd5753;
    6'd17: coeff = -17'sd15403;
    6'd18: coeff = -17'sd900;
    6'd19: coeff = -17'sd6800;
    6'd20: coeff = 17'sd3211;
    6'd21: coeff = -17'sd861;
    6'd22: coeff = 17'sd3083;
    6'd23: coeff = 17'sd466;
    6'd24: coeff = 17'sd827;
    6'd25: coeff = 17'sd0;
    6'd26: coeff = -17'sd395;
    6'd27: coeff = -17'sd106;
    6'd28: coeff = -17'sd338;
    6'd29: coeff = 17'sd49;
    6'd30: coeff = -17'sd111;
    default: coeff = 17'hXXX;

    //Band pass filter at [0.1,0.9] of Nyquist frequency
    /*5'd0:  coeff = 17'sd0;
    6'd1:  coeff = 17'sd256;
    6'd2:  coeff = 17'sd0;
    6'd3:  coeff = 17'sd344;
    6'd4:  coeff = 17'sd0;
```

```
      6'd5:  coeff = 17'sd0;
      6'd6:  coeff = 17'sd0;
      6'd7:  coeff = -17'sd1512;
      6'd8:  coeff = 17'sd0;
      6'd9:  coeff = -17'sd4525;
      6'd10: coeff = 17'sd0;
      6'd11: coeff = -17'sd8435;
      6'd12: coeff = 17'sd0;
      6'd13: coeff = -17'sd11810;
      6'd14: coeff = 17'sd0;
      6'd15: coeff = 17'sd52587;
      6'd16: coeff = 17'sd0;
      6'd17: coeff = -17'sd11810;
      6'd18: coeff = 17'sd0;
      6'd19: coeff = -17'sd8435;
      6'd20: coeff = 17'sd0;
      6'd21: coeff = -17'sd4525;
      6'd22: coeff = 17'sd0;
      6'd23: coeff = -17'sd1512;
      6'd24: coeff = 17'sd0;
      6'd25: coeff = 17'sd0;
      6'd26: coeff = 17'sd0;
      6'd27: coeff = 17'sd344;
      6'd28: coeff = 17'sd0;
      6'd29: coeff = -17'sd256;
      6'd30: coeff = 17'sd0;
      default: coeff = 17'hXXX;*/
   endcase
  end
endmodule

//////////////////////////////////////////////////////////////////////////////////
//
//servo module controls the servo that changes the camera location
//
//////////////////////////////////////////////////////////////////////////////////

module servo(clk, angle, pwm);
   input clk;
   input [8:0] angle;
   output reg pwm;

   reg [16:0] high_val;
```

```
    reg [16:0] low_val;
    reg servo_on;

    always @(posedge clk) begin
        if (servo_on == 0) begin
            high_val <= (angle/9)*1350 + 17'd25000;
            low_val <= 17'd87500 - high_val;
            servo_on <= 1;
        end

        if (high_val > 0) begin
            pwm <= 1;
            high_val <= high_val -1;
        end
        else if (low_val > 0) begin
            pwm <= 0;
            low_val <= low_val -1;
        end

        else begin
            servo_on <= 0;
        end
    end
endmodule




//////////////////////////////////////////////////////////////////////////////////
//
//angle_calculator module: calculates two angles: angle1 and angle2
//angle1: the angle created by the rays from the origin to the (x,y) location of the object
//and the positive x direction
//angle2: the angle created by the ray from the origin to the (x,y,z) location of the object
//with respect to the (x,y) plane.
//
//This module uses a lookup table initialized at reset to find the angles which minimizes
//the differences between x/y and tan(angle1) and between z/sqrt(x^2+y^2) and tan(angle2)
//
//Convention for angles:
//angle1 changes from 0 to 360 degrees
//angle2 changes from 0 at (x,y) = 0, z < 0 to 180 at (x,y) = 0, z > 0
//
```

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

module angle_calculator #(parameter M = 8)
   (input signed [M:0]  x_in,
   input signed [M:0]  y_in,
   input signed [M:0]  z_in,
   input clock,
   input calc_clock,
   output reg  [10:0] degree1,
   output reg  [10:0] degree2,
   output reg done1,
   output reg done2
   );

//    parameter [31:0] tan [0:84] = '{32'd18, 32'd36, 32'd54, 32'd72, 32'd90, 32'd108, 32'd128, 32'd144,
//       32'd162,  32'd181,  32'd199,  32'd218,  32'd236,  32'd255,  32'd274,  32'd294,  32'd313,  32'd333,
32'd353, 32'd373,
//       32'd393,  32'd414,  32'd435,  32'd456,  32'd477,  32'd499,  32'd522,  32'd544,  32'd568,  32'd591,
32'd615, 32'd640,
//       32'd665,  32'd691,  32'd717,  32'd744,  32'd772,  32'd800,  32'd829,  32'd859,  32'd890,  32'd922,
32'd955, 32'd989,
//       32'd1024,  32'd1060,  32'd1098,  32'd1137,  32'd1178,  32'd1220,  32'd1265,  32'd1311,  32'd1359,
32'd1409, 32'd1462,
//       32'd1518,  32'd1577,  32'd1639,  32'd1704,  32'd1774,  32'd1847,  32'd1926,  32'd2010,  32'd2100,
32'd2196, 32'd2300,
//       32'd2412,  32'd2534,  32'd2668,  32'd2813,  32'd2974,  32'd3152,  32'd3349,  32'd3571,  32'd3822,
32'd4107, 32'd4435,
//    32'd4818, 32'd5268, 32'd5807, 32'd6465, 32'd7286, 32'd8340, 32'd9743, 32'd11704};

   parameter sqroot = 5;
   parameter dnm = 1024;

   reg [31:0] tan [0:84];
   reg [M-1:0] x;
   reg [M-1:0] y;
   reg [M-1:0] z;
   reg [31:0] d;
   reg [31:0] nums;
   reg signx;
   reg signy;
   reg signz;
   reg done;
   reg control;
```

```
reg [63:0] squared;
reg back;

reg [7:0] count;
reg [7:0] index1;
reg [7:0] index2;
reg [31:0] min1;
reg [31:0] min2;

reg signed [31:0] temp1;
reg signed [31:0] temp2;

        wire signed [M:0] negx_in;
        wire signed [M:0] negy_in;
        wire signed [M:0] negz_in;

        assign negx_in = ~x_in + 1;       //absolute value of x (if x<0)
        assign negy_in = ~y_in + 1;       //absolute value of y (if y<0)
        assign negz_in = ~z_in + 1;       //absolute value of z (if z<0)

always @(posedge clock) begin
   if(calc_clock) begin
      tan [0] <= 2'd0;
      tan[1] <= 32'd18;
      tan[2] <= 32'd36;
      tan[3] <= 32'd54;
      tan[4] <= 32'd72;
      tan[5] <= 32'd90;
      tan[6] <= 32'd108;
      tan[7] <= 32'd128;
      tan[8] <= 32'd144;
      tan[9] <= 32'd162;
      tan[10] <= 32'd181;
      tan[11] <= 32'd199;
      tan[12] <= 32'd218;
      tan[13] <= 32'd236;
      tan[14] <= 32'd255;
      tan[15] <= 32'd274;
      tan[16] <= 32'd294;
      tan[17] <= 32'd313;
      tan[18] <= 32'd333;
      tan[19] <= 32'd353;
      tan[20] <= 32'd373;
```

```
tan[21] <= 32'd393;
tan[22] <= 32'd414;
tan[23] <= 32'd435;
tan[24] <= 32'd456;
tan[25] <= 32'd477;
tan[26] <= 32'd499;
tan[27] <= 32'd522;
tan[28] <= 32'd544;
tan[29] <= 32'd568;
tan[30] <= 32'd591;
tan[31] <= 32'd615;
tan[32] <= 32'd640;
tan[33] <= 32'd665;
tan[34] <= 32'd691;
tan[35] <= 32'd717;
tan[36] <= 32'd744;
tan[37] <= 32'd772;
tan[38] <= 32'd800;
tan[39] <= 32'd829;
tan[40] <= 32'd859;
tan[41] <= 32'd890;
tan[42] <= 32'd922;
tan[43] <= 32'd955;
tan[44] <= 32'd989;
tan[45] <= 32'd1024;
tan[46] <= 32'd1060;
tan[47] <= 32'd1098;
tan[48] <= 32'd1137;
tan[49] <= 32'd1178;
tan[50] <= 32'd1220;
tan[51] <= 32'd1265;
tan[52] <= 32'd1311;
tan[53] <= 32'd1359;
tan[54] <= 32'd1409;
tan[55] <= 32'd1462;
tan[56] <= 32'd1518;
tan[57] <= 32'd1577;
tan[58] <= 32'd1639;
tan[59] <= 32'd1704;
tan[60] <= 32'd1774;
tan[61] <= 32'd1847;
tan[62] <= 32'd1926;
tan[63] <= 32'd2010;
```

```
tan[64] <= 32'd2100;
tan[65] <= 32'd2196;
tan[66] <= 32'd2300;
tan[67] <= 32'd2412;
tan[68] <= 32'd2534;
tan[69] <= 32'd2668;
tan[70] <= 32'd2813;
tan[71] <= 32'd2974;
tan[72] <= 32'd3152;
tan[73] <= 32'd3349;
tan[74] <= 32'd3571;
tan[75] <= 32'd3822;
tan[76] <= 32'd4107;
tan[77] <= 32'd4435;
tan[78] <= 32'd4818;
tan[79] <= 32'd5268;
tan[80] <= 32'd5807;
tan[81] <= 32'd6465;
tan[82] <= 32'd7286;
tan[83] <= 32'd8340;
tan[84] <= 32'd9743;
tan[85] <= 32'd11704;
index1 <= 90;
min1 <= 32'd2147483640;
index2 <= 90;
min2 <= 32'd2147483640;

count <= 0;
nums <= 0;
done <= 0;
control <= 0;
done1 <= 0;
done2 <= 0;

                temp1 <= 32'd2147483640;
                        temp2 <= 32'd2147483640;
                        degree1 <= 0;
                        degree2 <= 0;

if( x_in < 0) begin
   signx <= 1;                  //1 means negative value
   x <= $unsigned(negx_in[M-1:0]);   //use the negative absolute value
end
```

```
    else begin
        signx <= 0;
    x <= $unsigned(x_in[M-1:0]);        //we can get rid of MSB because unsigned
    end

    if( y_in < 0) begin
       signy <= 1;
       y <= $unsigned(negy_in[M-1:0]);
    end

    else begin
        signy <= 0;
    y <= $unsigned(y_in[M-1:0]);
    end

    if( z_in < 0) begin
       signz <= 1;
       z <= $unsigned(negz_in[M-1:0]);
    end

    else begin
        signz <= 0;
    z <= $unsigned(z_in[M-1:0]);
    end

end

else if(~control & ~done) begin
    if(nums == 0) begin
       squared <= x * x + y * y;
       nums <= 1;
       back <= 0;
    end
    else begin
       if(~back && (nums * nums > squared)) back <= 1;
       else if(back && (nums * nums < squared)) begin
          control <= 1;
          d <= nums;
       end
       else if(back) nums <= nums - 1;
       else nums <= nums + sqroot ;
    end
end
```

```
      else if(control) begin
        if(count == 85) begin
          control <= 0;
          if(signz) degree2 <= 90 - index2;
                                          else degree2 <= 90 + index2;
          if(signx & signy) degree1 <= 180 + index1;
          else if(signx & ~signy) degree1 <= 180 - index1;
          else if(signy & ~signx) degree1 <= 360 - index1;
                                  else if(~signy & ~signx) degree1 <= index1;
          done1 <= 1;
          done2 <= 1;
          done <= 1;
        end
        else begin
          count <= count + 1;
          temp1 <= (y*dnm - tan[count]*x);
          temp2 <= (z*dnm - tan[count]*d);
          if((temp1 <= 0 && -1*(temp1) <= min1) || (temp1 > 0 && temp1 < min1)) begin
            if(temp1 > 0) min1 <= temp1;
            else min1 <= -1*temp1;
            index1 <= count + 1;
          end
          if((temp2 <= 0 && -1*(temp2) <= min2) || (temp2 > 0 && temp2 < min2)) begin
            if(temp2 > 0) min2 <= temp2;
            else min2 <= -1*temp2;
            index2 <= count + 1;
          end
        end
      end
    end
endmodule

//////////////////////////////////////////////////////////////////////////////////
//
//display_scale module displays scales on the x and y axis of the graph on the screen
//at inputted intervals. It is a powerful module as it enables the user to change
//the scaling in real time.
//
//////////////////////////////////////////////////////////////////////////////////

module display_scale
    #(parameter COLORY = 12'hF_F_F)
    (
```

```
 input clk,
 input reset,
 input [9:0] hcount,
 input [9:0] vcount,
 input [7:0] interval,
 output reg [3:0] pixel_R,
 output reg [3:0] pixel_G,
 output reg [3:0] pixel_B
 );

parameter [9:0] width = 10'd324;
parameter [9:0] length = 10'd240;
parameter linel = 3;
parameter lineh = 15;

reg [15:0] countx;
reg [15:0] county;
reg [15:0] countx2;
reg [15:0] county2;
reg [15:0] rcountx;
reg [15:0] rcounty;
reg done1;
reg done2;
reg condx;
reg condy;
reg switchx;
reg switchy;

always @(posedge clk) begin
    if(reset) begin
        countx <= 0;
        county <= 0;
        rcountx <= 0;
        rcounty <= 0;
        done1 <= 0;
        done2 <= 0;
        switchy <= 0;
        switchx <= 0;
    end
    else if(~done1 || ~done2) begin
        if(countx * interval < width) countx <= countx + 1;
        else begin
            rcountx <= countx - 1;
```

```
            done1 <= 1;
            countx2 <= countx - 1;
        end
        if(county * interval < length) county <= county + 1;
        else begin
            rcounty <= county - 1;
            done2 <= 1;
            county2 <= county - 1;
        end
    end
    else begin
            if(((vcount >= length - (lineh/2)) && (vcount <= length + (lineh/2))) && (((hcount >= (width + (countx2 * interval)) - (linel/2)) && (hcount <= (width + (countx2 * interval)) + (linel/2))) || ((hcount >= (width - (countx2 * interval)) - (linel/2)) && (hcount <= (width - (countx2 * interval)) + (linel/2))))) begin
            condx <= 1;
        end
        else if(condx && ((vcount >= length - (lineh/2)) && (vcount <= length + (lineh/2)))) begin
            condx <= 0;
            if(countx2 == 1 && ~switchx) switchx <= 1;
            if(countx2 == rcountx && switchx) switchx <= 0;
            if(~switchx) countx2 <= countx2 - 1;
            else countx2 <= countx2 + 1;
        end

            if(((hcount >= width - (lineh/2)) && (hcount <= width + (lineh/2))) && (((vcount >= (length + (county2 * interval)) - (linel/2)) && (vcount <= (length + (county2 * interval)) + (linel/2))) || ((vcount >= (length - (county2 * interval)) - (linel/2)) && (vcount <= (length - (county2 * interval)) + (linel/2))))) begin
            condy <= 1;
        end
        else if(condy && ((hcount >= width - (lineh/2)) && (hcount <= width + (lineh/2)))) begin
            condy <= 0;
            if(county2 == 1 && ~switchy) switchy <= 1;
            if(county2 == rcounty && switchy) switchy <= 0;
            if(~switchy) county2 <= county2 - 1;
            else county2 <= county2 + 1;
        end

        if((countx2 > 0) && (((vcount >= length - (lineh/2)) && (vcount <= length + (lineh/2))) && (((hcount >= (width + (countx2 * interval)) - (linel/2)) && (hcount <= (width + (countx2 * interval)) + (linel/2))) || ((hcount >= (width - (countx2 * interval)) - (linel/2)) && (hcount <= (width - (countx2 * interval)) + (linel/2)))))) begin
            pixel_R = COLORY[11:8];
            pixel_B = COLORY[7:4];
```

```
            pixel_G = COLORY[3:0];
         end
            else if((county2 > 0) && (((hcount >= width - (lineh/2)) && (hcount <= width + (lineh/2))) &&
(((vcount >= (length + (county2 * interval)) - (linel/2)) && (vcount <= (length + (county2 * interval)) +
(linel/2))) || ((vcount >= (length - (county2 * interval)) - (linel/2)) && (vcount <= (length - (county2 *
interval)) + (linel/2)))))) begin
            pixel_R = COLORY[11:8];
            pixel_B = COLORY[7:4];
            pixel_G = COLORY[3:0];
         end
         else begin
            pixel_R = 0;
            pixel_B = 0;
            pixel_G = 0;
         end
      end
   end
endmodule

//////////////////////////////////////////////////////////////////////////////
//
//display_angle module displays a moving line from the origin of the graph to the
//object's center on the graph
//
//////////////////////////////////////////////////////////////////////////////

module display_angle
   #(parameter COLORY = 12'hF_F_F)
   (input clk,
    input reset,
    input [9:0] hcount,
    input [9:0] vcount,
    input [10:0] x_display,
    input [10:0] y_display,
    output reg [3:0] pixel_R,
    output reg [3:0] pixel_G,
    output reg [3:0] pixel_B
    );

   parameter width = 320;
   parameter length = 240;
   parameter th = 35;
   parameter size = 12;
```

```
   reg signed [11:0] x;
   reg signed [11:0] y;
   reg signed [10:0] v;
   reg signed [10:0] h;
   reg signed [32:0] val;
   reg go;

   always @(posedge clk) begin
      v <= $signed({1'b0,vcount}) - length;
      h <= $signed({1'b0,hcount}) - width;
      if(reset) begin
         x <= 0;
         y <= 0;
         go <= 0;
      end
      else begin
         x <= $signed({1'b0,x_display}) - width + size;
         y <= $signed({1'b0,y_display}) - length + size;
         val <= v*x - h*y;
            if(((h > 0 && x > 0 && x > h) || (h < 0 && x < 0 && h > x)) && ((v > 0 && y > 0 && y > v) || (v < 0
&& y < 0 && v > y))) go <= 1;
         else go <= 0;

         if(go && ((val <= 0 && val > -1*th) || (val > 0 && val < th))) begin
            pixel_R = COLORY[11:8];
            pixel_B = COLORY[7:4];
            pixel_G = COLORY[3:0];
         end
         else begin
            pixel_R = 0;
            pixel_B = 0;
            pixel_G = 0;
         end
      end
    end
endmodule

//////////////////////////////////////////////////////////////////////////////////
//
//display_adjust module scales the input 65 signed bit x and y position of an object
//so that it fits the graph displayed on the screen. It keeps the object at the edge
//of the graph if the object tends to move out of the graph.
```

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

module display_adjust #(parameter xlowbit = 25, parameter xhighbit = 35, parameter ylowbit = 25, parameter yhighbit = 35)
 (input clock,
  input reset,
  input signed [64:0] x,
  input signed [64:0] y,
  output reg [10:0] x_display,
  output reg [10:0] y_display
  );

   parameter xshift = xhighbit - xlowbit + 1;
   parameter yshift = yhighbit - ylowbit + 1;

   parameter [15:0] width = 16'd320;
   parameter [15:0] length = 16'd240;

   parameter X = 6;
   parameter Y = 9;
   parameter WIDTH = 100;

   reg signed [11:0] dx;
   reg signed [11:0] dy;
   reg [63:0] tempx;
   reg [63:0] tempy;

   always @ (posedge clock) begin

      if(reset) begin
         dx <= $signed({x[64],x[xhighbit:xlowbit]});
         dy <= $signed({y[64],y[yhighbit:ylowbit]});
         tempx <= width;
         tempy <= length;
         y_display <= tempy[9:0] - Y;
         x_display <= tempx[9:0] - X;
      end
      else begin

                        if(x > 0 && $unsigned(x[63:0]) >= 64'h00000000_FFFFFFFF - WIDTH) dx <= $signed(12'b01_11111_11111 - WIDTH);
                else if(x < 0 && $unsigned(x[63:0]) <= 64'hFFFFFFFF_00000001 + 15*WIDTH/2) dx <=
```

```
$signed(12'b10_00000_00001 + 3*WIDTH/2);
        else dx = $signed({x[64],x[xhighbit:xlowbit]});

                        if(y > 0 && $unsigned(y[63:0]) >= 64'h00000000_FFFFFFFF - WIDTH) dy <=
$signed(12'b01_11111_11111 - WIDTH);
                else if(y < 0 && $unsigned(y[63:0]) <= 64'hFFFFFFFF_00000001 + 15*WIDTH/2) dy <=
$signed(12'b10_00000_00001 + WIDTH);
        else dy = $signed({y[64],y[yhighbit:ylowbit]});

        //dx = $signed({x[64],x[xhighbit:xlowbit]});
        //dy = $signed({y[64],y[yhighbit:ylowbit]});

          tempx <= (dx < 0) ? width - (($unsigned(-1*dx) * width) >> xshift) : (($unsigned(dx) * width) >>
xshift) + width;

          tempy <= (dy < 0) ? length + (($unsigned(-1*dy) * length) >> yshift)  : length - (($unsigned(dy) *
length) >> yshift);

        y_display <= tempy[9:0] - Y;
        x_display <= tempx[9:0] - X;
      end
   end

endmodule



//////////////////////////////////////////////////////////////////////////////////
//
// blob module displays a blob of given dimensions (in parameters) at a given location on the screen
//
//////////////////////////////////////////////////////////////////////////////////

module blob
  #(parameter WIDTH = 64,           // default width: 64 pixels
           HEIGHT = 64,          // default height: 64 pixels
           COLOR = 12'hF_F_F)  // default color: white
  (input [9:0] x_blob,hcount,
   input [9:0] y_blob,vcount,
   input clk,
   output reg [3:0] pix_R,
   output reg [3:0] pix_G,
   output reg [3:0] pix_B
```

```
  );

  always @(posedge clk) begin
        if ((hcount >= x_blob && hcount < (x_blob+WIDTH)) && (vcount >= y_blob && vcount < (y_blob+HEIGHT))) begin
            pix_R = COLOR[11:8];
            pix_B = COLOR[7:4];
            pix_G = COLOR[3:0];
          end

    else begin
      pix_R = 0;
      pix_B = 0;
      pix_G = 0;
    end

  end
endmodule


//////////////////////////////////////////////////////////////////////////////////
//
//adc_reading and adc_xyz modules are used to interface with the ADCs on the hardware board
//which digitize the analog signals from the accelerometer
//
//////////////////////////////////////////////////////////////////////////////////


module adc_reading (clk, read, write, xyz_mux, mux, ready, check2);
  input clk;
  output reg write;
  output reg read;
  output reg [1:0] mux;
  output reg [1:0] xyz_mux;
  output reg ready = 0;
  output reg check2;

  reg [11:0] time_counter = 0;
  reg [11:0] int_counter = 0;


  always @(posedge clk) begin
```

```
      if (time_counter < 12'b1111_1111_1111) begin
         ready <= 0;
         write <= 0;
         read <= 1;
         time_counter <= time_counter +1;

      end

      else begin
         write <= 1;
         read <=0;
         if (int_counter < 12'b1111_1111_1111) begin
            int_counter <= int_counter +1;
         end

         else begin
            ready <= 1;
            time_counter <= 0;
            int_counter <= 0;
            if (xyz_mux == 0) begin   //we were reading the x_value and now we switch to y_value
               xyz_mux <= 1;
               mux <= 1;

            end

            else if (xyz_mux == 1'd1) begin
               xyz_mux <= 2'b10;
               mux <= 2'b10;
               check2 <= 1;
            end

            else if (xyz_mux == 2'b10) begin
               xyz_mux <=0;
               mux <= 0;
               check2 <= 0;
            end

         end

      end

   end
endmodule
```

```
module adc_xyz (clk, ready, xyz_mux, coordinate, x_coor, y_coor, z_coor, check);

   input clk;
   input ready;
   input [1:0] xyz_mux;
   input [7:0] coordinate;


   output reg [7:0] x_coor;
   output reg [7:0] y_coor;
   output reg [7:0] z_coor;
   output reg check = 0;


   always @(posedge clk) begin
      if ((ready)) begin  //if we have a new value ready
         if (xyz_mux == 2'd1) begin   //if xyz_mux is high then we xy_mux was low so we have the x-value
            x_coor <= coordinate;
         end
         else if (xyz_mux == 0) begin
            z_coor <= coordinate;

         end
         else if (xyz_mux == 2'b10) begin
            y_coor <= coordinate;
            check <= 1;
         end
      end
   end

endmodule


// File:   cstringdisp.v
// Date:   24-Oct-05
// Author: I. Chuang, C. Terman
//
// Display an ASCII encoded character string in a video window at some
// specified x,y pixel location.
```

```
//
// INPUTS:
//
//   vclock      - video pixel clock
//   hcount      - horizontal (x) location of current pixel
//   vcount      - vertical (y) location of current pixel
//   cstring     - character string to display (8 bit ASCII for each char)
//   cx,cy       - pixel location (upper left corner) to display string at
//
// OUTPUT:
//
//   pixel       - video pixel value to display at current location
//
// PARAMETERS:
//
//   NCHAR       - number of characters in string to display
//   NCHAR_BITS   - number of bits to specify NCHAR
//
// pixel should be OR'ed (or XOR'ed) to your video data for display.
//
// Each character is 8x12, but pixels are doubled horizontally and vertically
// so fonts are magnified 2x.  On an XGA screen (1024x768) you can fit
// 64 x 32 such characters.
//
// Needs font_rom.v and font_rom.ngo
//
// For different fonts, you can change font_rom.  For different string
// display colors, change the assignment to cpixel.


//////////////////////////////////////////////////////////////////////////////
//
// video character string display: This module was taken from the 6.111 class website
//
//////////////////////////////////////////////////////////////////////////////

module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);

  parameter NCHAR = 8;             // number of 8-bit characters in cstring
  parameter NCHAR_BITS = 3; // number of bits in NCHAR

  input vclock;   // 65MHz clock
  input [10:0] hcount;    // horizontal index of current pixel (0..1023)
```

```
  input [9:0]      vcount; // vertical index of current pixel (0..767)
  output [2:0] pixel;       // char display's pixel
  input [NCHAR*8-1:0] cstring;   // character string to display
  input [10:0] cx;
  input [9:0]      cy;

  // 1 line x 8 character display (8 x 12 pixel-sized characters)

  wire [10:0]      hoff = hcount-1-cx;
  wire [9:0]       voff = vcount-cy;
  wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // < NCHAR
  wire [2:0]       h = hoff[3:1];          // 0 .. 7
  wire [3:0]       v = voff[4:1];                  // 0 .. 11

  // look up character to display (from character string)
  reg [7:0]  char;
  integer  n;
  always @(*)
    for (n=0 ; n<8 ; n = n+1 )                    // 8 bits per character (ASCII)
      char[n] <= cstring[column*8+n];

  // look up raster row from font rom
  wire reverse = char[7];
  wire [10:0] font_addr = char[6:0]*12 + v;    // 12 bytes per character
  wire [7:0]  font_byte;
  font_rom f(font_addr,vclock,font_byte);

  // generate character pixel if we're in the right h,v area
  wire [2:0] cpixel = (font_byte[7 - h] ^ reverse) ? 7 : 0;
  wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <= cx+NCHAR*16)
                        & (vcount < cy + 24));
  wire [2:0] pixel = dispflag ? cpixel : 0;

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// character_display module calculates the decimal numbers corresponding to the angle1, angle2, x, y and z corrdinates
// It calls char_string_display module to display the digits of the aforementioned decimal numbers
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module character_display #(parameter M = 12, A = 10)
   ( input reset,
     input clock,
     input [9:0] hcount,
     input [9:0] vcount,
     input signed [M:0] x,
     input signed [M:0] y,
     input signed [M:0] z,
     input [A:0] angle1,
     input [A:0] angle2,
     output [2:0] pixel
   );

   //Parameters
   parameter [15:0] width = 16'd640;
   parameter [15:0] length = 16'd480;

   parameter xstep = 8;
   parameter ystep = 12;
   parameter x0 = 50;
   parameter y0 = 3 * length / 4;

   parameter ts = 1000;
   parameter hs = 100;
   parameter tens = 10;

   //Wire declarations
   wire [2:0] pcx;
   wire [2:0] pcy;
   wire [2:0] pcz;
   wire [2:0] pca1;
   wire [2:0] pca2;
   wire [2:0] pwxval;
   wire [2:0] pwyval;
   wire [2:0] pwzval;
   wire [2:0] pwa1val;
   wire [2:0] pwa2val;
   wire [2:0] psx;
   wire [2:0] psy;
   wire [2:0] psz;

   wire [23:0] charx;
   wire [23:0] chary;
```

```
wire [23:0] charz;
wire [63:0] charangle1;
wire [63:0] charangle2;
wire [31:0] wxval;
wire [31:0] wyval;
wire [31:0] wzval;
wire [23:0] wa1val;
wire [23:0] wa2val;
wire [7:0] msignx;
wire [7:0] msigny;
wire [7:0] msignz;

wire [7:0] xvalts;
wire [7:0] xvalhs;
wire [7:0] xvaltens;
wire [7:0] xvalones;

wire [7:0] yvalts;
wire [7:0] yvalhs;
wire [7:0] yvaltens;
wire [7:0] yvalones;

wire [7:0] zvalts;
wire [7:0] zvalhs;
wire [7:0] zvaltens;
wire [7:0] zvalones;

wire [7:0] a1valhs;
wire [7:0] a1valtens;
wire [7:0] a1valones;

wire [7:0] a2valhs;
wire [7:0] a2valtens;
wire [7:0] a2valones;

wire [4:0] wxcts;
wire [4:0] wxchs;
wire [4:0] wxctens;
wire [4:0] wxcones;

wire [4:0] wycts;
wire [4:0] wychs;
wire [4:0] wyctens;
```

```
wire [4:0] wycones;

wire [4:0] wzcts;
wire [4:0] wzchs;
wire [4:0] wzctens;
wire [4:0] wzcones;

wire [4:0] wa1chs;
wire [4:0] wa1ctens;
wire [4:0] wa1cones;

wire [4:0] wa2chs;
wire [4:0] wa2ctens;
wire [4:0] wa2cones;

wire wminusx;
wire wminusy;
wire wminusz;

//Register declarations
reg [31:0] xval;
reg [31:0] yval;
reg [31:0] zval;
reg [23:0] a1val;
reg [23:0] a2val;

reg [2*M:0] posx;
reg [2*M:0] posy;
reg [2*M:0] posz;
reg [2*A:0] posa1;
reg [2*A:0] posa2;

reg [4:0] xcts;
reg [4:0] xchs;
reg [4:0] xctens;
reg [4:0] xcones;

reg [4:0] ycts;
reg [4:0] ychs;
reg [4:0] yctens;
reg [4:0] ycones;

reg [4:0] zcts;
```

```
reg [4:0] zchs;
reg [4:0] zctens;
reg [4:0] zcones;

reg [4:0] a1chs;
reg [4:0] a1ctens;
reg [4:0] a1cones;

reg [4:0] a2chs;
reg [4:0] a2ctens;
reg [4:0] a2cones;

reg minusx;
reg minusy;
reg minusz;

reg done;
reg started;

reg xctsdone;
reg xchsdone;
reg xctensdone;
reg xconesdone;

reg yctsdone;
reg ychsdone;
reg yctensdone;
reg yconesdone;

reg zctsdone;
reg zchsdone;
reg zctensdone;
reg zconesdone;

reg a1chsdone;
reg a1ctensdone;
reg a1conesdone;

reg a2chsdone;
reg a2ctensdone;
reg a2conesdone;

  char_string_display #(.NCHAR(3)) csd1(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pcx),
```

```
.cstring(charx), .cx(x0), .cy(y0));
     char_string_display #(.NCHAR(3)) csd2(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pcy), .cstring(chary), .cx(x0), .cy(y0 + (2*ystep)));
     char_string_display #(.NCHAR(3)) csd3(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pcz), .cstring(charz), .cx(x0), .cy(y0 + (4*ystep)));
    char_string_display #(.NCHAR(8)) csd4(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pca1), .cstring(charangle1), .cx(x0), .cy(y0 + (6*ystep)));
    char_string_display #(.NCHAR(8)) csd5(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pca2), .cstring(charangle2), .cx(x0), .cy(y0 + (8*ystep)));

     char_string_display #(.NCHAR(1)) csd6(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(psx), .cstring(msignx), .cx(x0 + (15*xstep)), .cy(y0));
     char_string_display #(.NCHAR(1)) csd7(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(psy), .cstring(msigny), .cx(x0 + (15*xstep)), .cy(y0 + (2*ystep)));
     char_string_display #(.NCHAR(1)) csd8(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(psz), .cstring(msignz), .cx(x0 + (15*xstep)), .cy(y0 + (4*ystep)));

   char_string_display #(.NCHAR(4)) csd9(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pwxval), .cstring(wxval), .cx(x0 + (17*xstep)), .cy(y0));
            char_string_display #(.NCHAR(4)) csd10(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pwyval), .cstring(wyval), .cx(x0 + (17*xstep)), .cy(y0 + (2*ystep)));
            char_string_display #(.NCHAR(4)) csd11(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pwzval), .cstring(wzval), .cx(x0 + (17*xstep)), .cy(y0 + (4*ystep)));
            char_string_display #(.NCHAR(3)) csd12(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pwa1val), .cstring(wa1val), .cx(x0 + (17*xstep)), .cy(y0 + (6*ystep)));
            char_string_display #(.NCHAR(3)) csd13(.vclock(clock), .hcount(hcount), .vcount(vcount), .pixel(pwa2val), .cstring(wa2val), .cx(x0 + (17*xstep)), .cy(y0 + (8*ystep)));

   assign pixel = pcx | pcy | pcz | pca1 | pca2 | pwxval | pwyval | pwzval | pwa1val | pwa2val | psx | psy | psz;
  assign charx = {8'd120, 8'd32, 8'd61};
  assign chary = {8'd121, 8'd32, 8'd61};
  assign charz = {8'd122, 8'd32, 8'd61};
  assign charangle1 = {8'd97,8'd110,8'd103,8'd108,8'd101,8'd49, 8'd32, 8'd61};
  assign charangle2 = {8'd97,8'd110,8'd103,8'd108,8'd101,8'd50, 8'd32, 8'd61};
  assign wxval = xval;
  assign wyval = yval;
  assign wzval = zval;
  assign wa1val = a1val;
  assign wa2val = a2val;

  assign wxcts = xcts;
  assign wxchs = xchs;
```

```
assign wxctens = xctens;
assign wxcones = xcones;

assign wycts = ycts;
assign wychs = ychs;
assign wyctens = yctens;
assign wycones = ycones;

assign wzcts = zcts;
assign wzchs = zchs;
assign wzctens = zctens;
assign wzcones = zcones;

assign wa1chs = a1chs;
assign wa1ctens = a1ctens;
assign wa1cones = a1cones;

assign wa2chs = a2chs;
assign wa2ctens = a2ctens;
assign wa2cones = a2cones;

assign wminusx = minusx;
assign wminusy = minusy;
assign wminusz = minusz;

assign msignx = wminusx ? 8'd45 : 8'd32;
assign msigny = wminusy ? 8'd45 : 8'd32;
assign msignz = wminusz ? 8'd45 : 8'd32;

ascii_number anxcts(.index(wxcts),.coeff(xvalts));
ascii_number anxchs(.index(wxchs),.coeff(xvalhs));
ascii_number anxctens(.index(wxctens),.coeff(xvaltens));
ascii_number anxcones(.index(wxcones),.coeff(xvalones));

ascii_number anycts(.index(wycts),.coeff(yvalts));
ascii_number anychs(.index(wychs),.coeff(yvalhs));
ascii_number anyctens(.index(wyctens),.coeff(yvaltens));
ascii_number anycones(.index(wycones),.coeff(yvalones));

ascii_number anzcts(.index(wzcts),.coeff(zvalts));
ascii_number anzchs(.index(wzchs),.coeff(zvalhs));
ascii_number anzctens(.index(wzctens),.coeff(zvaltens));
ascii_number anzcones(.index(wzcones),.coeff(zvalones));
```

```
ascii_number ana1chs(.index(wa1chs),.coeff(a1valhs));
ascii_number ana1ctens(.index(wa1ctens),.coeff(a1valtens));
ascii_number ana1cones(.index(wa1cones),.coeff(a1valones));

ascii_number ana2chs(.index(wa2chs),.coeff(a2valhs));
ascii_number ana2ctens(.index(wa2ctens),.coeff(a2valtens));
ascii_number ana2cones(.index(wa2cones),.coeff(a2valones));

always @(posedge clock) begin
   if(reset) begin
      done <= 1;
      started <= 0;
      minusx <= 0;
      minusy <= 0;
      minusz <= 0;
      xval <= {8'd48, 8'd48, 8'd48, 8'd48};
      yval <= {8'd48, 8'd48, 8'd48, 8'd48};
      zval <= {8'd48, 8'd48, 8'd48, 8'd48};
      a1val <= {8'd48, 8'd48, 8'd48};
      a2val <= {8'd48, 8'd48, 8'd48};
   end

   if(done) begin

      started <= 1;
      done <= 0;
      if(x < 0) begin
         minusx <= 1;
         posx <= $unsigned(-1*x);
      end
      else begin
         minusx <= 0;
         posx <= $unsigned(x);
      end

      if(y < 0) begin
         minusy <= 1;
         posy <= $unsigned(-1*y);
      end
      else begin
         minusy <= 0;
         posy <= $unsigned(y);
```

```
end

if(z < 0) begin
   minusz <= 1;
   posz <= $unsigned(-1*z);
end
else begin
   minusz <= 0;
   posz <= $unsigned(z);
end

posa1 <= angle1;
posa2 <= angle2;

xcts <= 0;
xchs <= 0;
xctens <= 0;
xcones <= 0;

ycts <= 0;
ychs <= 0;
yctens <= 0;
ycones <= 0;

zcts <= 0;
zchs <= 0;
zctens <= 0;
zcones <= 0;

a1chs <= 0;
a1ctens <= 0;
a1cones <= 0;

a2chs <= 0;
a2ctens <= 0;
a2cones <= 0;

xctsdone <= 0;
xchsdone <= 0;
xctensdone <= 0;
xconesdone <= 0;

yctsdone <= 0;
```

```
    ychsdone <= 0;
    yctensdone <= 0;
    yconesdone <= 0;

    zctsdone <= 0;
    zchsdone <= 0;
    zctensdone <= 0;
    zconesdone <= 0;

    a1chsdone <= 0;
    a1ctensdone <= 0;
    a1conesdone <= 0;

    a2chsdone <= 0;
    a2ctensdone <= 0;
    a2conesdone <= 0;

  end
  else if(started) begin
    if(~xctsdone) begin
      if(xcts * ts <= posx) xcts <= xcts + 1;
      else begin
        xcts <= xcts - 1;
        posx <= posx - (xcts - 1) * ts;
        xctsdone <= 1;
      end
    end
    if(xctsdone && ~xchsdone) begin
      if(xchs * hs <= posx) xchs <= xchs + 1;
      else begin
        xchs <= xchs - 1;
        posx <= posx - (xchs - 1) * hs;
        xchsdone <= 1;
      end
    end
    if(xchsdone && ~xctensdone) begin
      if(xctens * tens <= posx) xctens <= xctens + 1;
      else begin
        xctens <= xctens - 1;
        posx <= posx - (xctens - 1) * tens;
        xctensdone <= 1;
      end
    end
```

```
if(xctensdone && ~xconesdone) begin
   if(xcones <= posx) xcones <= xcones + 1;
   else begin
      xcones <= xcones - 1;
      posx <= posx - (xcones - 1);
      xconesdone <= 1;
   end
end


if(~yctsdone) begin
   if(ycts * ts <= posy) ycts <= ycts + 1;
   else begin
      ycts <= ycts - 1;
      posy <= posy - (ycts - 1) * ts;
      yctsdone <= 1;
   end
end
if(yctsdone && ~ychsdone) begin
   if(ychs * hs <= posy) ychs <= ychs + 1;
   else begin
      ychs <= ychs - 1;
      posy <= posy - (ychs - 1) * hs;
      ychsdone <= 1;
   end
end
if(ychsdone && ~yctensdone) begin
   if(yctens * tens <= posy) yctens <= yctens + 1;
   else begin
      yctens <= yctens - 1;
      posy <= posy - (yctens - 1) * tens;
      yctensdone <= 1;
   end
end
if(yctensdone && ~yconesdone) begin
   if(ycones <= posy) ycones <= ycones + 1;
   else begin
      ycones <= ycones - 1;
      posy <= posy - (ycones - 1);
      yconesdone <= 1;
   end
end
```

```
if(~zctsdone) begin
   if(zcts * ts <= posz) zcts <= zcts + 1;
   else begin
      zcts <= zcts - 1;
      posz <= posz - (zcts - 1) * ts;
      zctsdone <= 1;
   end
end
if(zctsdone && ~zchsdone) begin
   if(zchs * hs <= posz) zchs <= zchs + 1;
   else begin
      zchs <= zchs - 1;
      posz <= posz - (zchs - 1) * hs;
      zchsdone <= 1;
   end
end
if(zchsdone && ~zctensdone) begin
   if(zctens * tens <= posz) zctens <= zctens + 1;
   else begin
      zctens <= zctens - 1;
      posz <= posz - (zctens - 1) * tens;
      zctensdone <= 1;
   end
end
if(zctensdone && ~zconesdone) begin
   if(zcones <= posz) zcones <= zcones + 1;
   else begin
      zcones <= zcones - 1;
      posz <= posz - (zcones - 1);
      zconesdone <= 1;
   end
end

if(~a1chsdone) begin
   if(a1chs * hs <= posa1) a1chs <= a1chs + 1;
   else begin
      a1chs <= a1chs - 1;
      posa1 <= posa1 - (a1chs - 1) * hs;
      a1chsdone <= 1;
   end
end
if(a1chsdone && ~a1ctensdone) begin
```

```
   if(a1ctens * tens <= posa1) a1ctens <= a1ctens + 1;
   else begin
      a1ctens <= a1ctens - 1;
      posa1 <= posa1 - (a1ctens - 1) * tens;
      a1ctensdone <= 1;
   end
end
if(a1ctensdone && ~a1conesdone) begin
   if(a1cones <= posa1) a1cones <= a1cones + 1;
   else begin
      a1cones <= a1cones - 1;
      posa1 <= posa1 - (a1cones - 1);
      a1conesdone <= 1;
   end
end

if(~a2chsdone) begin
   if(a2chs * hs <= posa2) a2chs <= a2chs + 1;
   else begin
      a2chs <= a2chs - 1;
      posa2 <= posa2 - (a2chs - 1) * hs;
      a2chsdone <= 1;
   end
end
if(a2chsdone && ~a2ctensdone) begin
   if(a2ctens * tens <= posa2) a2ctens <= a2ctens + 1;
   else begin
      a2ctens <= a2ctens - 1;
      posa2 <= posa2 - (a2ctens - 1) * tens;
      a2ctensdone <= 1;
   end
end
if(a2ctensdone && ~a2conesdone) begin
   if(a2cones <= posa2) a2cones <= a2cones + 1;
   else begin
         a2cones <= a2cones - 1;
      posa2 <= posa2 - (a2cones - 1);
      a2conesdone <= 1;
   end
end

if(xconesdone && yconesdone && zconesdone && a1conesdone && a2conesdone) begin
```

```
            xval <= {xvalts, xvalhs, xvaltens, xvalones};
            yval <= {yvalts, yvalhs, yvaltens, yvalones};
            zval <= {zvalts, zvalhs, zvaltens, zvalones};
            a1val <= {a1valhs, a1valtens, a1valones};
            a2val <= {a2valhs, a2valtens, a2valones};
            started <= 0;
            done <= 1;
          end
      end
   end

endmodule

////////////////////////////////////////////////////////////////////////////////
//
//ascii_number module serves as a look-up table mapping digits to their ascii numbers.
//
////////////////////////////////////////////////////////////////////////////////

module ascii_number(
  input wire [4:0] index,
  output reg [7:0] coeff
   );
  always @(index) begin
    case (index)
      5'd0:  coeff = 8'd48;
      5'd1:  coeff = 8'd49;
      5'd2:  coeff = 8'd50;
      5'd3:  coeff = 8'd51;
      5'd4:  coeff = 8'd52;
      5'd5:  coeff = 8'd53;
      5'd6:  coeff = 8'd54;
      5'd7:  coeff = 8'd55;
      5'd8:  coeff = 8'd56;
      5'd9:  coeff = 8'd57;
      default: coeff = 8'hXX;
    endcase
   end
endmodule
```