Automatic Projector Tilt Compensation System

Ganesh Ajjanagadde Shantanu Jain James Thomas

October 11, 2015

Abstract

We designed a system that corrects the input to a projector if it is tilted so that its output appears unskewed. We read input from a NTSC (National Television System Committee) video camera and store it in an internal block memory. We then process the frame stored in memory using a perspective transformation to pre-warp the image that is sent to the projector via a VGA (Video Graphics Array) signal. The parameters of the perspective transformation are obtained from an accelerometer, which senses two axes of tilt. This allows automatic keystone correction in the two directions sensed by the accelerometer provided the output screen is vertical. Our method also includes options for manual keystone correction to any degree desired, for any projector and screen orientations. For ease of manual correction, we provide the option of using a test pattern (a checkerboard). We also play some useful audio for the percentage of pixels kept by the transformation.

Acknowledgments

First and foremost, we would like to thank our 6.111 instructor Gim Hom for his tremendous patience, experience, and intuition regarding digital systems. This project would never have been possible without him, and countless number of times he saved our group a huge amount of time by some very crucial observations. Not only that, he also played a very important role in our choice of project topic. Initially, we were planning to create some sort of Bitcoin miner. I think it is safe to say that we are all glad that we chose this topic instead on his suggestion.

We would also like to thank all the teaching assistants, lab assistants, and CI-M writing instructors for their very useful suggestions and feedback. We particularly appreciate the guidance of one of our TA's José E. Cruz Serallés for his wealth of experience with Verilog and Xilinx's tools. His project on recursive augmented reality was also very useful as a reference point for certain aspects, such as generation of clocks of different frequencies and the correct timing of sync and blank signals for a 640 × 480 @60 Hz VGA display. Furthermore, he was able to identify a subtle bug with our memory address computation.

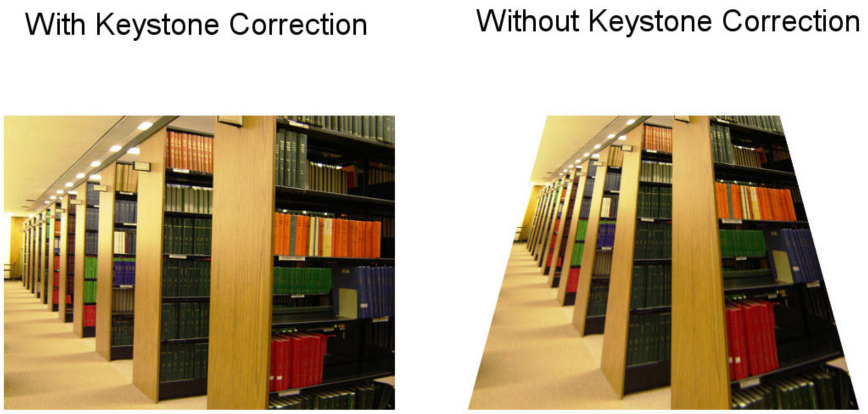
Last, but not least, we would like to thank all our peers in this class. Their thoughtful questions during our proposal presentation raised issues that we had not anticipated. Furthermore, they often were willing to help with some commonly faced issues in the lab, thereby avoiding duplication of effort. We particularly appreciate the note on Piazza (our online discussion forum) by Andres Erbsen regarding test benches and use of Icarus Verilog.

Contents

1 Introduction

Due to the advances in semiconductor technology, today's display projectors can incorporate fairly sophisticated digital processing algorithms for various enhancements to the visual appearance. Moreover, there is an increasing prevalence of portable projectors that benefit from fast, automated setup. One desired functionality is keystone/tilt correction. In other words, even when the projector is tilted, the projector should be able to "pre-warp" the image so that its output appears unskewed. Figure 1 shows what is meant by the keystone effect and what its desired correction should look like [1].

Figure 1: Keystone Correction In Vertical Direction



In this project, we project the output of a camera, connect the camera's output to the FPGA (Field Programmable Gate Array) board, and the FPGA board's VGA output to the projector. We mount an accelerometer on the projector and measure its signals to determine the projector's tilt angle on two axes. We then run a perspective transformation algorithm on the FPGA that warps the camera output based on the tilt angles and produces the results at the VGA output for the projector to display. We also provide a manual correction mode so that any desired correction can be achieved. This manual correction mode is exposed to the user via the arrow keys and switches on the FPGA kit. This is of use in mainly two cases:

- The automatic correction obtained using the accelerometer readings is unsatisfactory or inadequate.
- The user desires to correct for projector orientation in the third axis, or in the case when the screen is non-vertical.

[1] https://en.wikipedia.org/wiki/File:Vertical-keystone.jpg

Finally, we also provide a useful voice output for the percentage of pixels kept after the perspective transformation (this is a lossy transformation in general). The audio is triggered by pressing a button on the FPGA kit.

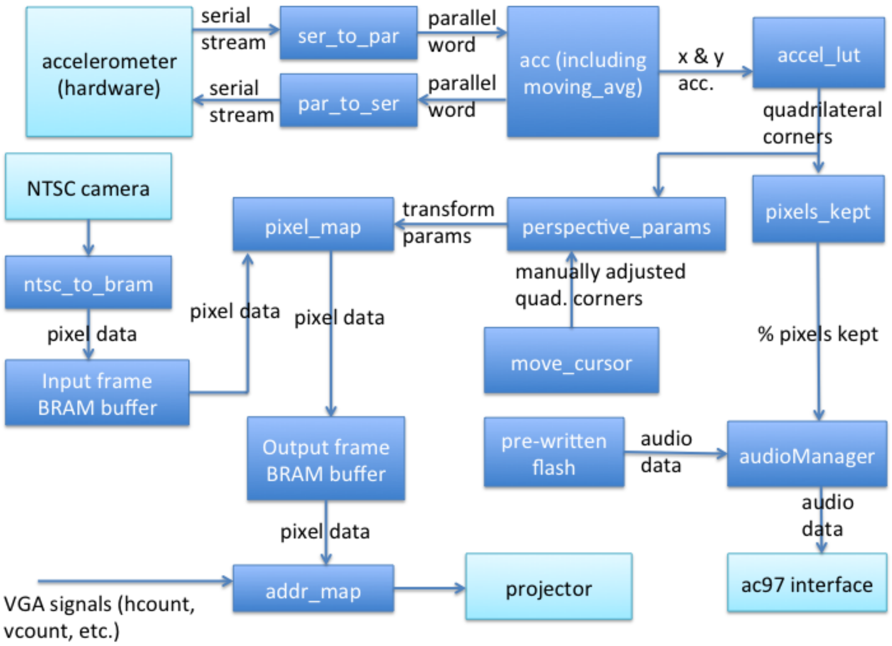
2 Previous Work

Given the practical importance of keystone correction, there has been significant research on doing automatic keystone correction. While Raskar and Beardsley [2001] and Sukthankar et al. [2001] provide a solution to this problem, their methods suffer from one principal weakness, already noted in Li and Sezan [2004], namely that their algorithms are not well suited for implementation in an embedded system. For instance, their methods require solving an 8x8 system of linear equations via Gaussian elimination or similar methods, which is not suitable for embedded platforms, particularly those with slow dividers. Our key technical contribution is the elimination of the Gaussian elimination algorithm, and instead replacing it with closed form solutions for the equations. Moreover, we implement our algorithm on a 6 million gate Xilinx Virtex 2 FPGA, demonstrating that our algorithm is suitable for embedded systems use. The work done in Li and Sezan [2004] subsumes most of this work. However, their system is much more complex, as evidenced by the use of a camera for computing the precise perspective transformation. For the scope of this term final project, we use a simpler accelerometer based approach instead, paying the price of loss of automatic correction along one axis. Note that we still retain fully general manual correction ability.

3 Module Architecture

Figure 2 shows a block diagram including all of the major modules in our system, which are explained further below and in later sections:

Figure 2: Block Diagram



Our keystone correction system can be divided into roughly four functional components:

1. Accelerometer interface
2. Perspective transformation
3. I/O (Input/Output) interface
4. Audio system

Each functional component handles a specific collection of tasks, and is further broken down into a set of one or more modules.

3.1 Accelerometer interface

The accelerometer presents a SPI interface for data transfer. The par_to_ser module converts a multi-bit value to a serial stream of bits for transfer to the accelerometer. The ser_to_par module converts a serial stream of bits read from the accelerometer to a multi-bit value of desired width. The moving_avg module computes the average of 32 accelerometer readings. This is done to counteract some of the noise in the accelerometer readings. The acc module uses the par_to_ser and ser_to_par modules to initialize the accelerometer and fetch x and y acceleration readings from it in a loop, averaging them using the moving_avg module.

3.2 Perspective transformation

The perspective transformation component is where all core computations are performed. The accel_lut module accepts the accelerometer readings from the accelerometer as an index into a ROM (read-only memory) containing coordinates of four coordinates of a quadrilateral. The corners of the quadrilateral (after multiplexing with manual override parameters) are fed into the perspective_params module, which computes necessary perspective transformation parameters. The perspective parameters are then fed into pixel_map, which maps the screen rectangle onto the quadrilateral described above via a perspective transformation. The pixels_kept module is a side module that accepts the corners of the quadrilateral, and computes the percentage of pixels kept during the transformation. This is of use in the audio system.

3.3 I/O (Input/Output) interface

The input/output interface to various buttons/switches is contained in the top-level labkit.v file. Here, the hardware of the labkit is mapped to the modules, and the functional components are connected together. The relevant buttons/switches are passed to move_cursor module, which manipulates the corners of the quadrilateral when the user wishes to manually adjust the correction. The bram module implements a very simple 2 port block memory on the FPGA. Two instances of this are created, one called ntsc_buf storing the camera input, and one called vga_buf containing the processed output (from the perspective transformation). Essentially, we have a pipeline formed: input camera writes to ntsc_buf, perspective transformation reads from ntsc_buf and writes to vga_buf, and the output VGA signal is formed by looping over the vga_buf.

3.4 Audio system

The audio system is responsible for receiving, saving, and playing back audio. This requires interacting with several different pieces of hardware and external modules. The top-level module of this functional component is audioManager. When triggered by a switch setting, audioManager prepares itself for receiving the audio tracks as a continuous stream of data from the external USB input hardware. Because of the difficult-to-implement timing requirements of the external hardware, we use the 6.111 staff provided usb_input module to simplify receiving data over USB. As audio data is being received over USB, it is written to the labkit's on-board flash memory. Flash memory is also difficult to use, since a time-sensitive start up sequence needs to be followed to enable proper use of the flash hardware. We made use of the staff provided flash_manager, test_fsm, and flash_int modules. This set of modules greatly abstracts away the low-level details of working with the flash. When the audio is triggered via an external button input, audioManager handles queuing up the appropriate set of tracks for sequential playback. The percentage of pixels used is an external input to audioManager from the pixels_kept module.

4 Design Decisions

There are several noteworthy design decisions.

4.1 Use of 640×480 @60 Hz VGA

The first of them is the use of a 640×480 @60 Hz VGA display. The use of 640×480 as opposed to the more common higher resolutions found today is due to the memory and computational limitations of our FPGA kit. Initially, we hoped that we could get away with generic code that does not explicitly tie in heavily with the specific screen resolution. Unfortunately, this is not easy, since the choice of resolution influences many different aspects. Chief among these are the sizes of memory vs available memory trade-off, the bit widths of x and y coordinates, the size of the accel_lut ROM, and the bit widths of numerous other quantities such as the dividers and the multipliers in pixel_map and perspective_params respectively. The choice of 60 Hz was made on the basis of its almost guaranteed availability: almost all VGA displays support this refresh rate.

4.2 Use of NTSC Camera as Input

Also related to input/output is our decision to use an NTSC camera feed as the input to the FPGA. Ideally, we would have liked to hook up a computer's VGA signal to the FPGA, so that we can demonstrate the correction system in a more realistic setting. Unfortunately, the labkit in this course has only a single VGA port, ruling out this option.

4.3 Choice of Memory Architecture

Another major design decision made is the choice of memory architecture. We initially planned on using the available 36 Mbits of ZBT memory spread across 2 banks. This would allow us to store at least 4 frames at full 640×480 resolution and 24 bit color (8 bit R, 8 bit G, 8 bit B). Unfortunately, it turns out that ZBT memory is not dual-ported, so read and write on the same bank can't be achieved simultaneously. Since the perspective transform (operating pixel by pixel) turns out to be a huge computational bottleneck, we did not want to waste additional time waiting for read/write on ZBT. Moreover, coordination between the memory banks storing processed and unprocessed data would require some sort of arbiter, adding complexity to our project. Thus, we chose instead to go with the block memory on the FPGA. This can be made into true dual-port memory, and has a single read/write cycle latency, as opposed to the multi-cycle latency of ZBT memory. However, the amount of block memory available on the FPGA is only 2.5 Mbits. This required sacrifice on image quality. We chose a combination of image down-sampling and color depth reduction. More specifically, we chose a 320x240 sized memory, with 12 bits per line (4 bit R, 4 bit G, 4 bit B). This results in approximately 1 Mbit per buffer, and we use 2 buffers, leaving us with nearly 512k for the accel_lut ROM. That is more than sufficient for our accel_lut. It is certainly possible to squeeze some more color depth, e.g a 14 bit asymmetrical division among R, G, and B. This should help the image quality, but our visual tests indicated no substantial improvement, and hence we omitted this.

4.4 Choice of clocks

A technically noteworthy design decision is our choice of clocks. To avoid clock domain issues, as much as possible we used multiples of a common system clock. Using the Xilinx DCM (Digital Clock Manager), we synthesized a 50 MHz clock from the labkit's standard 27 MHz clock. We used this as a global sys_clk. 640×480 @60Hz needs to be driven at nearly 25 MHz, and was thus obtained by multiplying the time period of sys_clk by 2. The NTSC camera must be driven at around 27 MHz, and the appropriate clock is already generated by the labkit - clock_27mhz. To avoid change of the perspective parameters mid-frame, we also needed a slow_clk signal, i.e one who's time period is on the order of seconds. This can't be accomplished through the use of DCM that easily. Moreover, since actual timing violations for this signal are not really that important, we implemented a simple "flip clock on reaching a count" style "clock divider".

The audio system used the native clock_27mhz for all logic excluding providing data to the AC97 audio codec hardware. The operation of providing audio samples to the AC97 operated on the AC97's external clock. This is required because the AC97 plays back audio at a 48kHz rate, a common audio sampling rate. On spare cycles between the AC97's external clock and the native clock, the system performed all other operations, including track queueing and track switching.

4.5 User interface considerations

An important factor in the quality of the projected image is the percentage of pixels used. The more correction is applied to the image, the more image quality suffers and the dimmer the resulting projection. We wanted to provide the user of the system this information. They could act on this feedback by further adjusting the projector if possible. We realized only power users would be able to make use of this feedback, so we wanted to provide it in a non-intrusive manner - audio was the most natural candidate. The decision to use audio was not without drawbacks. Hearing-impaired users would unfortunately not be able to make use of the feedback. Since the labkit's on-board display was already used for the manual correction system, there was no other way to display the percentage of pixels used without showing it on the projected screen, which we had earlier eliminated considering it was only useful to power users.

The manual adjustment functionality relies on the user being able to select the corner to be manipulated via a set of switches. While this binary representation would be natural to computer engineers and mathematicians, it would not be self-explanatory to non-technical users. We were again constrained by the labkit's limited hardware input devices from making a more natural interface. Of course, in theory we could have hooked up an external keyboard to another one of the labkit's ports, but this would have increased the complexity of our project.

5 Module Descriptions

In this section, we give a more detailed description of each module, along with the respective contributors.

5.1 par_to_ser (James)

This module converts a word of some size W (e.g. reg[W-1:0]) to a stream of bits, one at each clock cycle, to be sent to the accelerometer. The higher-order bits are produced first. This module is necessitated by the SPI communication protocol of the accelerometer.

5.2 ser_to_par (James)

This module converts a stream of bits received from the accelerometer, one per clock cycle, to a word of desired width. It expects that the higher-order bits will be received first.

5.3 moving_avg (James)

This module uses a ring buffer to store the last 32 acceleration readings received from the accelerometer for a particular dimension. When a new sample is received, the oldest sample is removed, the sum of the last 32 samples is adjusted accordingly, and the average is recomputed. Since we compute the average over 32 (2^5) samples, we do not need dividers and can use bit shifts for the needed divisions. This module is meant to smooth acceleration readings, which are expected to fluctuate slightly even when the accelerometer is held fixed.

5.4 acc (James)

The timing diagram for 4-wire SPI interaction with the accelerometer is shown below:

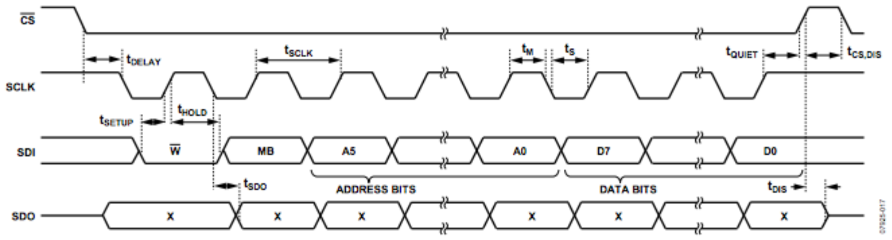


Figure 5. SPI 4-Wire Write

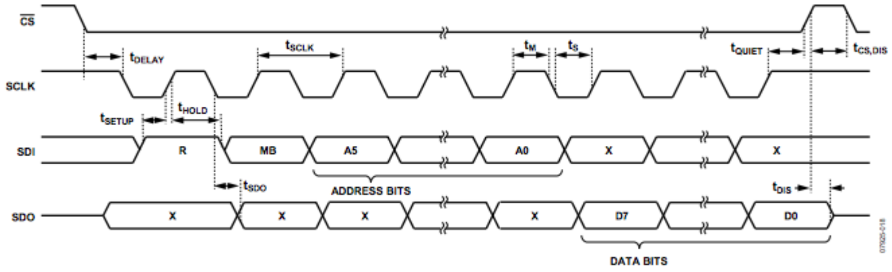


Figure 6. SPI 4-Wire Read

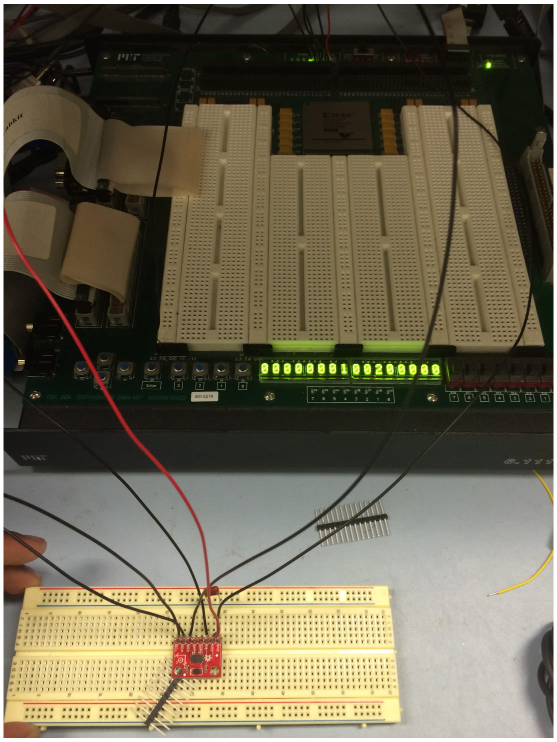
This module first initializes the accelerometer and then reads x and y acceleration values from it in a loop. The clock used for accelerometer communications is our 50 MHz system clock slowed by a factor of 20, which meets the the max clock frequency spec of 5 MHz. Most of the default configurations for the accelerometer are acceptable for our purposes; the only initialization we need to do is set the measure bit of the POWER_CTL register (each register contains a single byte). Since this register is at address 0x2D, and since we want to perform a single-byte write to it, we first drive the SDI pin with the serial bit stream 8'b00101101 (see Figure 5 in the timing diagram upper two bits for mode and lower six bits for register address). This specifies the mode and register address, and can then be used to drive it with the data to be stored in the register. By default, the accelerometer can measure acceleration values in the range of $\pm 2g$. This is sufficient for our purposes since we care only about static acceleration (which is in the range $\pm 1g$).

After 16 accelerometer clock cycles have passed, we know that we have sent all of the necessary initialization data to the accelerometer. We then transition our accelerometer state machine to the x-y read loop. Since the x and y acceleration readings are 16 bits (two registers' worth, since the accelerometer gives us 10 bits of precision), we want to perform a multi-byte read of the x and y registers. This can be done in the manner shown in Figure 6 of the timing diagram. We simply assert the R and MB bits, continue driving SDI with the address bits for the first x or first y register (the second registers for x and y immediately follow the first, which is why the multi-byte read of contiguous registers is acceptable), and then wait 16 additional cycles for all of our data to be produced on the SDO line. We rotate between a state for reading the x registers and a state for reading the y registers. We feed the produced x and y readings into instantiations of the moving_avg filter. The output of the acc module is the output of these filters.

One challenge we faced was getting the accelerometer clock period correct. Even though the spec states that the maximum clock frequency is 5 MHz, if we use a 5 MHz clock and then use a single clock cycle to implement the $t_{CS,DIS}$ waiting period shown in Figures 5 and 6 of the timing diagram, we will not meet the minimum $t_{CS,DIS}$ spec of 250ns. So we had to pick a clock frequency lower than $\frac{1s}{250ns} = 4$ MHz to use a one-cycle wait to meet the $t_{CS,DIS}$ spec. This bug took quite a while to find.

The picture below shows the accelerometer module working in isolation, with averaged accelerometer readings displayed on the hex display:

Figure 3: Accelerometer readings on hex display



5.5 accel_lut (Ganesh)

The accel_lut module provides a look-up table from the accelerometer readings in two directions to the four corners of the quadrilateral. The output is synchronized to the global sys_clk. The module is essentially implemented as a giant case statement, from which Xilinx's tools are able to infer a ROM of the appropriate size. Although the data coming out of the accelerometer has 10 bits of precision in each axis, using the full precision would require too much space for the ROM. Instead, we use 6 bits for each of the 2 axes, for a total of 12 bits as an index into the ROM. Each word of the ROM is 76 bits wide. This is because each x coordinate is 10 bits wide (0 to 639), each y coordinate is 9 bits wide (0 to 479), and there are 4 corners. Since each entry corresponds to a choice of quadrilateral corners, and there are $2^{12} = 4096$ entries, manual entry of the look-up table is infeasible. Instead, we wrote a code generator accel_lut.jl in the Julia programming language (`julialang.org`). Essentially, the code generator accepts an input CSV (comma separated values) file containing some entries of the desired look-up table obtained via manual testing. The code generator then does a mathematical interpolation to obtain the remaining values of the look-up table, and writes out the accel_lut.v file. Of course, it may be argued that the interpolation is not sufficient to get fine grained accuracy. Indeed, our demonstration did show that our look-up table could use some refinement. However, we believe that this is not an inherent deficiency of the approach; it just means that we need more data points and sufficient polynomial degree of the interpolator. Also related to this is another merit to this method: as the user obtains more and more entries, he/she can simply populate the CSV file with them. The quality of interpolation can then only improve.

5.6 pixels_kept (Ganesh)

The pixels_kept module accepts the four corners of the quadrilateral, computes its area, and expresses it as a percentage of the total area. The area formula of a quadrilateral in terms of the coordinates of its four corners is a simple determinant expansion, and may be found easily in elementary geometry references. For a quadrilateral with coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$, its area A is given by the expression:

$$2A = |(x_1 - x_3)(y_2 - y_4) - (y_1 - y_3)(x_2 - x_4)|. \tag{1}$$

The division by 640×480, and subsequent multiplication by 100 is accomplished without doing either a hardware division or multiplication for efficiency reasons. The key observation here is that $64 \times 48 = 2^{10} \times 3$. Thus, it suffices to figure out how to divide by 3 without performing a hardware division. For this, we approximate 3 by $\frac{21}{64}$, giving us the result correct to the nearest percent. The output is then a 7 bit value expressing the percentage from 0 to 100.

5.7 bram (Ganesh)

The bram module is a very simple true dual port memory module, with exactly enough storage for a down-sampled frame (320x240) with 12 bit color depth per pixel. By convention, the first port is always a write port, triggered by a WE (write enable) signal. The second port is a read port. The ports have their own individual address lines and clock signals in order to achieve the dual ported-ness. The advantages of a dual port memory over a single port memory were clear in our case, and have already been outlined. One more advantage of a dual port memory is that the two ports can be driven at different clocks. The only place where the dual ported abstraction breaks down is when one tries to read and write to the same address simultaneously. It is clear that there is no reasonable behavior in that case. Our design takes this into account, and guarantees that at most one pixel in each frame stores a garbage value. Given the large fraction of displays with at least one defective pixel, this is an extremely minor issue. We initially used Xilinx's proprietary IP Coregen application to synthesize the BRAM (block random access memory) module. However, in order to maximize portability and minimize the use of proprietary files, we studied this a little more and found out that with the appropriate Verilog code, Xilinx's tools will automatically infer the presence of a BRAM. This allowed us to use our own very simple BRAM implementation. It also gave us one additional benefit: Coregen modules often take significantly longer to synthesize as compared to inferred ones.

5.8 addr_map (Ganesh)

The addr_map is a critical single-line module that abstracts away memory address computations, allowing one to address the correct location in memory by providing the x coordinate and y coordinate locations. This is a pure combinational logic module, with no synchronization to any clock. Essentially, it takes in an x coordinate in $[0, 639]$ and a y coordinate in $[0, 479]$, and computes the memory address in the 320x240 line BRAM buffer. This is achieved by taking the integer part of a division by 2 of the x and y coordinates, followed by a standard mapping of a 2 dimensional matrix address to a flat array, an exact analog of manipulating a matrix that has been heap allocated in the C programming language. We unfortunately had a somewhat subtle bug in our memory address computation. At its core, it boils down to the fact that integer division by two followed by a multiplication is not the same as multiplication followed by integer division by two. Fortunately, we tested this early on, and noticed a checkerboard pattern with horizontal stripes on the VGA display instead of a neat checkerboard. The credit for finding that bug goes to our TA José.

5.9 slow_clk (Ganesh, James)

The slow_clk is a simple module that takes in a high frequency clock signal (on the order of MHz) and generates a signal with a much lower frequency.

For us, this was of use in generating a clock with a frequency on the order of a Hz for synchronizing the quadrilateral corner locations and perspective transform parameters. It was also of use in generating a clock signal that met the manufacturer specs for the accelerometer (see the acc section). It is implemented by having a looping counter that would result in an inversion of the clock signal every time the counter hit a certain number of "ticks". Note that this method is not a robust way of generating an in-phase clock with no skew. It is, however, quite well suited for the task of generating much lower frequency clocks, such as a 1000 times lower frequency. For other applications, such as dividing a clock frequency by a factor of 2, we used Xilinx's DCM (Digital Clock Manager) instead.

5.10 move_cursor (Ganesh)

The move_cursor module is a user interface module for manipulating the four corners of a quadrilateral. It allows one to move the four corners of a quadrilateral by pressing the four arrow keys and selecting the corner index through the switches on the labkit. To avoid accidental manual adjustment of keystone correction, the movement can only be done when an override switch is on. This module is very similar to the controls used to move the paddle in the Pong game that we designed for Lab 3, and no tricky debugging was needed for this.

5.11 perspective_params (Ganesh)

The perspective_params module lies at the very heart of this project. It accepts a list of the four corners of the quadrilateral $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ and finds the parameters $p_i, 1 \leq i \leq 9$ such that the general perspective transformation:

$$(x, y) \to \left(\frac{p_1 x + p_2 y + p_3}{p_7 x + p_8 y + p_9}, \frac{p_4 x + p_5 y + p_6}{p_7 x + p_8 y + p_9} \right) \tag{2}$$

maps the outer coordinates of the screen $(0, 0), (0, 480), (640, 480), (640, 0)$ onto $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ respectively. The module also computes the coefficients $pinv_i$ of the inverse mapping, which is also a perspective transformation (due to the algebraic group structure of perspective transformations which can be checked by direct calculation). The reason for computing the $pinv_i$ is mainly due to our initial goal of having a "memory-less" output transformation, in the sense that given the desired VGA coordinate, we can compute the pre-image of that point. This in turn was due to an inadequate understanding of the FPGA memory and VGA behavior. For instance, we assumed that the VGA frame rate can be controlled simply by adjusting the VGA clock, an assumption that is incorrect. This lack of understanding at one point almost threatened the successful completion of our project, and thus we strongly recommend future 6.111 students to think and discuss very hard the hardware limitations before starting actual implementation work. We were fortunately able to come up with the BRAM design, which although suffers from poorer image quality, nevertheless demonstrates the soundness of our algorithm. It turns out that the BRAM design can work equally well with either p_i or $pinv_i$. However, at this point of the project, we had already started using $pinv_i$ and did not want to take the risk of going back to p_i. Computing both p_i and $pinv_i$ takes up approximately 80 % of the 144 available 18x18 bit signed multipliers on the FPGA. This may be easily reduced to around 20 % by eliminating the computation of $pinv_i$, something we would have done with additional time for the project.

Note that the p_i (or $pinv_i$) may be scaled by an arbitrary constant, so without loss of generality, we assumed that $p_3 = 1$. However, to avoid needless divisions, in the actual solutions implemented on the FPGA, we scale all parameters to ensure that they are all integers. The closed-form solution to the set of equations (2) (as implemented on the FPGA) is given below:

$$\begin{aligned}
p_7 &= 3[(x_1 - x_4)(y_2 - y_3) + (y_1 - y_4)(x_3 - x_2)] && (3)\\
p_8 &= 4[(x_1 = x_2)(y_3 - y_4) + (x_4 - x_3)(y_1 - y_2)] && (4)\\
d &= x_4(y_2 - y_3) + x_2(y_3 - y_4) + x_3(y_4 - y_2) && (5)\\
p_9 &= 1920d && (6)\\
p_3 &= 1920x_1d && (7)\\
p_6 &= 1920y_1d && (8)\\
p_1 &= x_4p_7 + 3(x_4 - x_1)d && (9)\\
p_2 &= x_2p_8 + 4(x_2 - x_1)d && (10)\\
p_4 &= y_4p_7 + 3(y_4 - y_1)d && (11)\\
p_5 &= y_2p_8 + 4(y_2 - y_1)d && (12)\\
pinv_1 &= p_6p_8 - p_5p_9 && (13)\\
pinv_2 &= p_2p_9 - p_3p_8 && (14)\\
pinv_3 &= p_3p_5 - p_2p_6 && (15)\\
pinv_4 &= p_4p_9 - p_6p_7 && (16)\\
pinv_5 &= p_3p_7 - p_1p_9 && (17)\\
pinv_6 &= p_1p_6 - p_3p_4 && (18)\\
pinv_7 &= p_5p_7 - p_4p_8 && (19)\\
pinv_8 &= p_1p_8 - p_2p_7 && (20)\\
pinv_9 &= p_2p_4 - p_1p_5 && (21)\\
d_x &= 639pinv_1 && (22)\\
d_y &= 639pinv_4 && (23)\\
d_d &= 639pinv_7 && (24)
\end{aligned}$$

d_x, d_y, d_d are a couple of parameters used to avoid multiplications in the actual mapping described by the perspective transformation (2). More precisely, they allow clients of this module (such as pixel_map in our case) to simply execute a two dimensional loop over the image, incrementing/decrementing the numerator and denominator on each iteration as opposed to performing a fresh multiplication. Essentially, they correspond to the decrements needed at the end of each horizontal scan line. Note the use of 480, 640 (i.e $\frac{1920}{4}$ and $\frac{1920}{4}$) as opposed to the more precise 479, 639. The reason for this is that 480, 640 are multiples of sizable powers of two, reducing the bit width needed for certain multiplications required in the solution. Moreover, the difference caused by this slight error can be ignored, due to the continuity of the perspective transformation. Quite crucial to the ease of solving this set of equations symbolically by hand was a good choice of basis. We were fortunate that the "natural basis" in terms of the screen coordinates is actually a pretty good one in that sense.

The outputs of this module are synchronized on a slow_clk signal. The rationale for this is two fold:

1. Due to large number of multiplications, even with pipe-lining, it is unlikely that we can meet timing requirements of sys_clk (50 MHz).

2. It is undesirable to change the parameters mid-frame anyway, and having a huge latency in changing these parameters may thus in fact be desirable.

5.12 pixel_map (Ganesh)

The pixel_map module uses the parameters $pinv_i$ and d_x, d_y, d_d obtained in the perspective_params module to do the actual perspective transformation pixel by pixel. At a high level, it is doing a simple loop through x and y through $[0, 639]$ and $[0, 479]$ respectively, computing

$$\left(\frac{pinv_1 x + pinv_2 y + pinv_3}{pinv_7 x + pinv_8 y + pinv_9}, \frac{pinv_4 x + pinv_5 y + pinv_6}{pinv_7 x + pinv_8 y + pinv_9}\right)$$

at each x and y. It then uses the results to obtain the memory address via addr_map in the ntsc buffer, and uses another addr_map to write the data found in the ntsc buffer to the VGA buffer. When the coordinates of the inverse transformation are out of range, the module writes a black pixel. A state machine is used to start the divider and keep track of the general state of the computation. The divider used is based on the restoring division algorithm, and was provided by the staff. There was a subtle bug in the staff-provided divider, resulting in incomplete divisions being written out. For us, that translated into always reading an address of 0 of the ntsc buffer, leading to a uniform color over the whole frame. Essentially, the bug boiled down to an insufficient width for a counter register in the divider. Tracking down the bug was pretty difficult, since it first required verifying that our pixel_map module was providing the correct numerator and denominator. A test bench (runnable under the free software Icarus Verilog simulator) confirmed that there was something wrong with the divider. Another test bench then verified that the results of the division were indeed incorrect. Finally, a close examination of the divider code resulted in the identification of the bug. Once this bug was fixed, the module worked correctly. Unfortunately, the divider needs a width of 79 bits, and the staff provided divider is not pipe-lined. Xilinx's IP Coregen provides pipe-lined dividers up to a width of 32 bits, which is insufficient for our needs. Furthermore, they discourage creation of pipe-lined dividers of greater width due to the high area cost. As such, we require 80 clock cycles per pixel, resulting in a frame rate of 1-2 frames per second. Note that with sufficient hardware resources, this may be easily converted into a real-time system.

5.13 ntsc_to_bram (James)

Most of the staff code from the zbt_6111 example was kept intact, including the ADV7185 initialization module and the NTSC decoding module. ntsc_to_zbt was converted to ntsc_to_bram, which involved a few changes. First off, ROW_START and COL_START were changed to be 0, since we want our image to fill the entire screen (as opposed to the staff example, where the upper left corner of the image does not start at pixel (0,0)). We expanded all of the registers holding NTSC data to have widths of 30 bits (as opposed to 8), since we want 12-bit color and need the full YCrCb data. We maintained the original approach of reading only scan lines from field 0 and alternately writing them to even and odd screen rows. Since we stored pixels in BRAM, which has a maximum capacity of 2.5 Mbits, and we needed to store a frame from the camera and another frame with the transformed output in addition to the look-up table (discussed above), we only had space to store a 320 by 240 frame with 12 bits of color per pixel. So we cropped the NTSC input by taking only the upper left 320 by 240 rectangle.

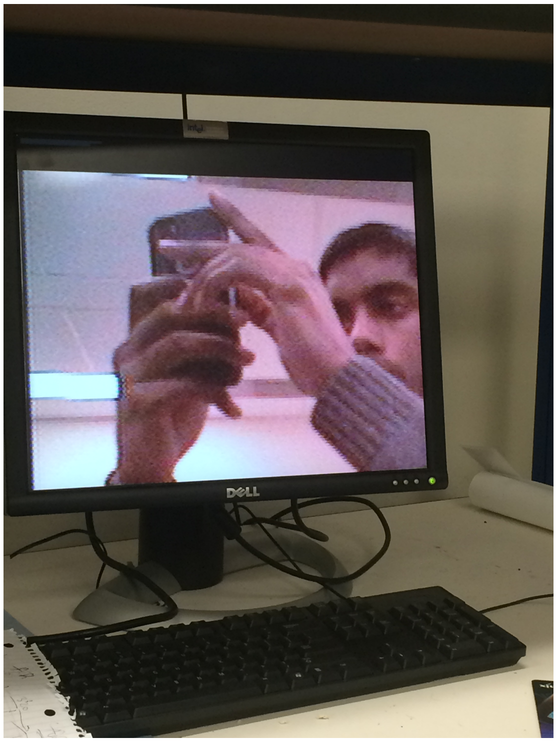
Since we now store one pixel per memory location as opposed to four (we were able to size our BRAM so that each location was 12 bits), we write to BRAM whenever the x address is less than 320 and the y address is less than 240 and we are at a write enable positive edge, given by the we_edge signal in the code. (In the staff code, we wrote to ZBT only when the x address was a multiple of 4 since we stored 4 pixels per memory location). We determined the BRAM address from the x and y coordinates by treating BRAM as a 2D array with 240 rows and 320 columns laid out in row-major order (row 1 in the first 320 locations, row 2 in the next 320, etc.). So we simply multiplied the y coordinate by 320 and added the x coordinate to get the BRAM address. Finally, since we needed to convert the YCrCb data to RGB using the staff-provided module, which takes 3 clock cycles, we needed to delay the BRAM address and BRAM write enable signals by three clock cycles as well using the staff-provided synchronize module so that we wrote the right data to the right addresses.

Since our BRAM has both read and write ports, we were able to write camera data to the BRAM at the same time that our transformation code fetched pixels from this BRAM, eliminating the need for any coordination mechanisms.

One major challenge in designing this module was eliminating the appearance of randomly colored pixels in the video output. We expected the video to be grainy because we were scaling a 320 by 240 image to 640 by 480 resolution for projection, but the random pixels were unexplained until Gim pointed out that we were writing the BRAM at a different clock rate (the system clock rate) than the NTSC data was coming to us (the tv_in_line_clock1 rate). This was causing some setup times at the BRAM to be violated when clocks were out of phase and the BRAM to be written with unstable values.

Below is an image of the ntsc_to_bram code working in isolation:

Figure 4: camera output using bram



5.14 audioManager (Shantanu)

The audioManager module is the core module in the audio playback component. It handles the whole life-cycle of audio, from receiving over USB, to storing into flash, queuing tracks, and playback. It interfaces with three external hardware components - the flash chip, the USB interface, and the AC97 audio codec hardware.

The module is centered around its interaction with the flash hardware, since the flash memory is manipulated any time the module is active. Depending on the input of the switches, it either sets the flash memory to "read mode" or "write mode." To prevent the module from resetting the flash memory and deleting its contents, which the labkit automatically does when it is powered on, the module also has a “reset disable” switch.

When in “write mode”, the labkit prepares itself to receive data from the usb hardware, via the usb_input module provided by the staff. Whenever usb_input indicates a new data sample is available, the “dowrite” input to the staff provided flash_manager module is asserted to be true, in order to trigger the write operation. The system continues in this cycle until the user’s computer indicates all data has been transmitted, at which time he must exit write mode by manipulating the switches to “read mode.” Due to the limitations of flash_manager, data must be written in one session - the user does not have the ability to leave “write mode,” analyze the data on the flash memory, and re-enter “write mode” to continue writing, without first resetting the flash and deleting all the data it contains. As a result, this module inherits these limitations, and so a user may have to manually disable the “reset disable” switch to trigger a reset if his data transfer was to fail. When in “read mode” the flash memory’s contents can be retrived one location at a time, by setting the read address parameter “raddr”. We keep this value at address 0, until we receive the signal to begin audio playback. Based on the percentage of pixels kept, an input signal to the module, the sequence of audio tracks to be played is decided. For example, if eighty-nine percent of pixels are used, four audio tracks are queued for playback: “Eighty,” “Nine,” “Percent,” “Used.” In particular, the number eighty nine is constructed out of two separate sequential tracks instead of a single recording. This system allows us to save memory, reducing transfer time and accumulated error, at the expense of additional complexity in the module.

Having decided what tracks are to be played back, playback begins by calculating the address of the currently playing track, and retriving the contents of that memory location. While many schemes may be used to calculate the location of the track, we made our recording so that every track is exactly one second long. We then used a simple multiplication of one second of samples by a track index to calculate the track address. Based on the AC97’s external 48kHz clock, we provided a new audio sample to the AC97 to acheive audio playback. After one second of samples had been played back, we calculated the address of the next track queued and began playback. A special end of playback marker called “UNUSED_TRACK” halted playback.

5.15 BCD (Shantanu)

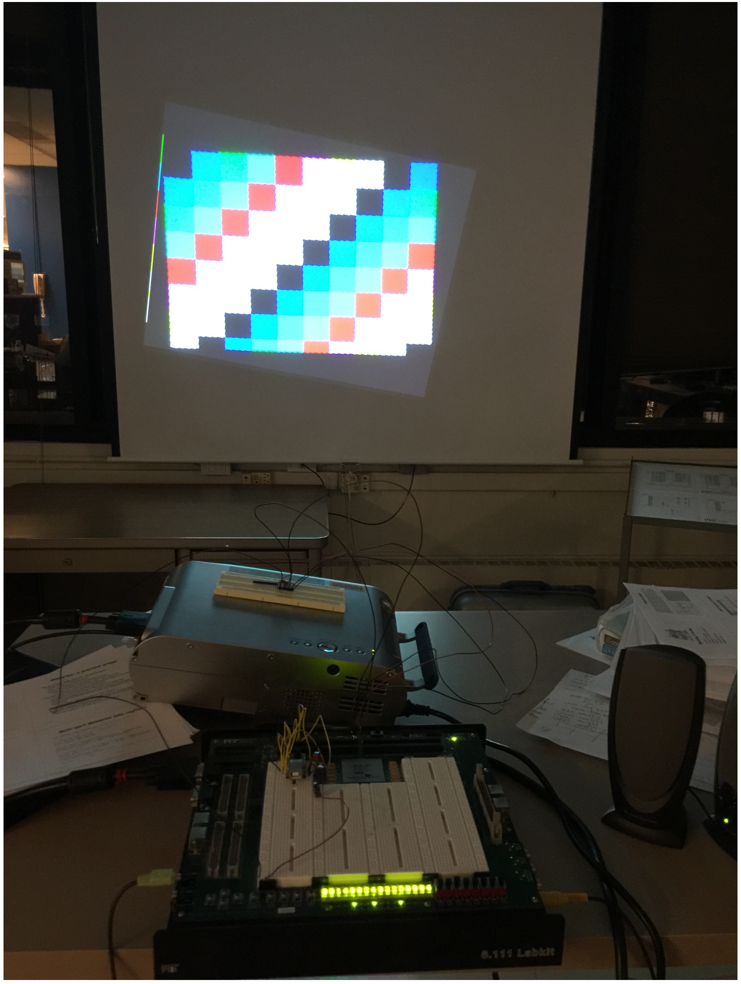
This module converts from a binary representation to a decimal representation. It was used within the audioManager module to convert the percentage of pixels kept from binary to a decimal representation in order to assign the tracks to be played. This was necessary since digital systems naturally use binary representations, but our audio output was necessarily in decimal.

Technically, we opted to implement this module as a look-up table that assigned the decimal ones and tens place based on the input value. The alternative was to use a computational algorithm implemented in Verilog, which was widely available. We used the look-up table because we thought it provided more clarity to the user than the seemingly-opaque efficient algorithms, at the expense of utilizing more of the scarce look-up tables on the hardware.

6 Final Product and User Interface

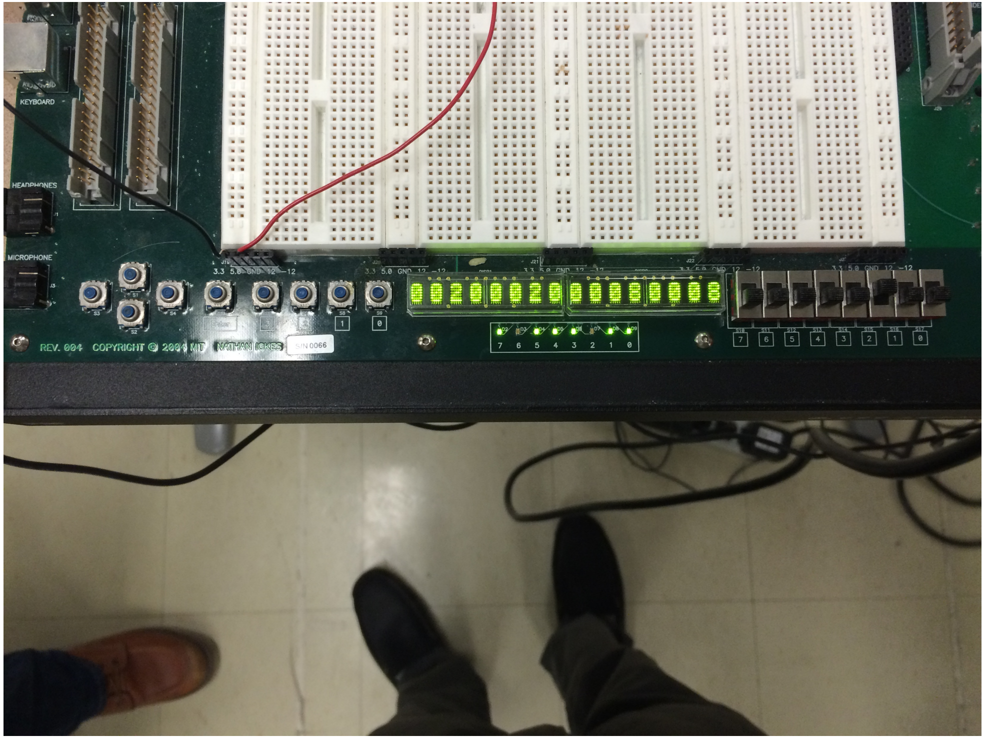
Below is an image of our working system automatically correcting the input to a tilted projector so that it appears rectangular:

Figure 5: Automatic Keystone Correction



Below is an image of the user interface (UI) to our correction system:

Figure 6: User Interface



When pressed, button 0 triggers the audio system, which announces the percentage of screen pixels currently used for the corrected image. LEDs 7-1 show the binary representation of this value. Buttons 2 and 3 adjust the volume of the audio system up or down. The first four digits on the hex display show the current x acceleration value, the next four show the current y acceleration value, the next four show the x coordinate of the corner of the quadrilateral selected by switches 0 and 1, and the last four show the y coordinate of this corner. If switch 7 is high, we are in manual correction mode, and the accelerometer readings are ignored. In this mode, we can manually adjust the corner of the quadrilateral selected by switches 0 and 1 by pressing the up and down buttons (to adjust the x coordinate) and the left and right buttons (to adjust the y coordinate). When switch 5 is high, we display a checkerboard on the screen instead of the NTSC camera input.

7 Testing And Debugging

Testing and debugging Verilog-defined hardware is complicated due to the large synthesis times. Unlike software, where one can change a single line of code, and trigger an incremental build that can complete on the order of seconds, synthesis of FPGA hardware is a lengthy process, taking on the order of minutes or even hours depending on the amount of logic. Our complete avoidance of Coregen modules definitely served us well in this respect. We also consciously often commented out unnecessary modules. This provided dual benefits:

- Much shorter synthesis times
- Ensuring that the bug is being isolated as much as possible.

Nevertheless, these two steps are certainly not sufficient in efficient debugging. As such, we adopted a multi-pronged testing and debugging approach. Broadly speaking, we used the following methods:

- Icarus Verilog test benches/verification
- Xilinx ModelSim test benches/verification
- Julia implementations/tests of algorithms
- Labkit I/O, e.g led's, hex display, logic analyzer probes
- Staring at code/data-sheets
- Integration tests

7.1 Icarus Verilog test benches/verification

By far the most effective and efficient way of testing simple modules in our experience was the use of Icarus Verilog. We are extremely grateful to a fellow student Andres for posting a note on our online discussion forum (Piazza) regarding the benefits and use of Icarus Verilog. Defining and using test benches through Icarus Verilog is extremely easy and efficient. In our experience, this was most useful for verifying long and complicated chains of combinational logic, such as mathematical algorithms. It was also very useful for checking the syntax of our Verilog code. As an example of its utility, we found a bug with perspective_params that originated from incorrect mixed use of signed and unsigned arithmetic. The main drawbacks of Icarus Verilog are its lack of visualizations of waveforms when invoked on the command line, and its decreasing utility with complex state machines and other sequential logic. The first of these drawbacks is addressed quite well by Xilinx ModelSim.

7.2 Xilinx ModelSim test benches/verification

ModelSim helped us to discover a bug in the staff-provided divider module. We instantiated a 79-bit divider and were not getting correct results. When we examined waveforms in ModelSim, we saw that the ready bit was being asserted far fewer than 79 cycles after we started the division. Upon closer examination of the module, it turned out that the counter the module was using to keep track of the current bit was only 6 bits wide, which supports only operand bit widths up to 63. We extended the width of this counter to 7 bits to solve this issue.

ModelSim was also used to verify the functionality of the BCD module. After uncovering a bug in the percentage of pixels kept during integration, the first area of investigation was the BCD module, since it was the least tested module. It was trivial to write a test bench for the module and verify the output was correct using ModelSim.

7.3 Julia implementations/tests of algorithms

Prior to implementing the perspective transforms on the FPGA, we definitely wanted to verify the soundness of our approach in software first. We settled on the use of the Julia programming language for this purpose, though in all likelihood MATLAB or Python(+numpy/scipy/matplotlib) would have served just as well. This was by far the most effective means of testing whether an algorithm is correct or not, since one can rely on the benefits of fast debug/iterate cycles in software. This is particularly true in the case of languages featuring a REPL (read, evaluate, print loop) based interpreter, such as Julia. Correcting code is as simple as making some changes, re-including the file in the REPL, and rerunning. This helped us particularly in the verification of the pixels_kept and perspective_params module. As an aside, to avoid using too many languages for software implementations, we used Julia for our code generator for the accel_lut as well. As a concrete example of a serious bug caught using the help of our software implementations, we were able to track and correct an error in the computation of p_1 and p_2 using this method. Essentially, the bug was an interchange of two terms in the closed form expressions that occurred while transcribing the closed form solutions we found for p_1 and p_2 from paper to the computer code.

7.4 Labkit I/O

We wired up the logic analyzer to the accelerometer pins to verify that the acc module was correctly initializing the accelerometer (e.g. the par_to_ser module was producing the correct serial bit stream for the register address and data). It turned out that the bug in the accelerometer was elsewhere (see the next section).

7.5 Staring at code/data-sheets

This is definitely a questionable inclusion, and we do not recommend this method in general. Nevertheless, it is often effective when done correctly and in the right spirit, since it forces one to rethink and step through the logic again, questioning all assumptions. For instance, once we confirmed that the divider implementation was incorrect, we had to examine the divider code line by line.

It did not take us too long to realize the mistake in the implementation, and looking back we do not see any other way of correcting the mistake.

As described in the section on the acc module, taking a second pass through the data-sheet for the accelerometer helped us to identify an issue with a violated timing constraint ($t_{CS,DIS}$).

7.6 USB input to flash memory issues

The most challenging issue in the audio component of the project was receiving data over USB and storing the samples into flash memory. When we received data over USB, we wrote it immediately to the flash memory. However, after a data transfer, we noticed that many of the bytes were missing.

The flash memory is already difficult to work with because of its tight timing constraints. It is also an old piece of hardware, with read and write times considered slow at the time of manufacture of the labkit over ten years ago. We theorized that the flash memory was too slow to be able to write data at the rate it was being received.

To isolate the device at fault, a series of unit tests were developed. One test took data from a look-up table coded into the device via Verilog and placed it into the flash memory. When we analyzed the flash memory contents, we found it was exactly the same as the look-up table. From this test we were able to conclude that the flash memory was functional. We developed another test that counted the number of bytes received over the USB module before writing them to flash. We found that while the number of bytes received was exactly the number of bytes transmitted from the computer, the number of bytes in memory was still off by a factor between two and three. Even while buffering the data from the USB input device, itself also a buffer, we could not improve the performance of the system.

Despite repeatedly consulting with instructor Gim Hom, teaching assistant Luis Fernandez (who previously implemented a working audio system), and the author of the staff-provided flash_manager module, and in spite of our best efforts to understand and modify the code, we could not improve the performance enough to record all samples. We did improve our system so that the audio was intelligible. With Gim's approval, we closed this issue as not resolvable in the scope of our project and moved forward.

7.7 Integration tests

In addition to testing modules in isolation, we obviously also needed to test whether the modules work together or not. Fortunately, the audio system could be quite easily separated from the rest of the system, so we anticipated easy integration here. Ironically, we faced a strange problem here, which we never understood. The issue was with the percentage value computed by pixels_kept. Initially, pixels_kept was actually named pixels_lost, and was computing 100 minus the percentage of pixels_kept. The audio module was written for percentage of pixels kept however, so initially the audio module did a second subtraction from 100 to get the correct value. The playback of the percentage was often incorrect for some reason (never identified). Initially, we were keen on fixing the issue from the audio end, since pixels_lost was a known correct module. However, we had no luck fixing it from the audio end, so instead we renamed pixels_lost to pixels_kept and removed the subtraction from 100. This somehow fixed the issue.

Much easier was the integration with the accelerometer. This went flawlessly after we agreed upon the clock frequencies used by the accelerometer and the system. We made sure that the clocks used by the accelerometer and the accel_lut (to which the accelerometer readings are sent) were integer multiples of one another to prevent any setup or hold time violations when clocks were out of phase.

Integration of the transformation logic with the memory interface was also extremely smooth, apart from the subtle bug in address computation uncovered by our TA José.

One of the most tedious aspects was the collection of data values for the look-up table. It took around 2 hours of fiddling with the manual correction to get 12 readings for doing the interpolation. Lacking the appropriate measurement equipment, it is difficult to verify that the vertical edges of the checkerboard are exactly parallel to the vertical edges of the screen. Furthermore, we also needed to ensure that the 4:3 aspect ratio was not distorted.

8 Future Work

Although our projector tilt compensation system met most of our initial goals, there are a wide variety of extensions and modifications to the project that we would have liked to implement.

On the image side, we would like to take some steps to improve the quality of the projected image. As noted earlier, one of our key design decisions was the use of reduced resolution (320 × 240) and reduced color depth (12 bit as opposed to 24 bit). This can be achieved with either a labkit with more dual-ported memory, or our existing labkit with a sophisticated ZBT based memory design (with an arbiter). Another improvement that can be made to the images is the use of some sort of anti-aliasing filter. For instance, when we compute the coordinates in the pixel_map module, we only look at the integer part of the division (corresponding to a single pixel index). By looking at the fractional part, we could interpolate the color values at neighboring pixels and use that to reduce the "jagged edges" associated with aliasing in images.

Another significant unknown is how well the accelerometer approach works when the projector is not kept at a fixed distance from the screen. We have not tested whether this would affect the quality of the automatic correction. If this does affect quality, we are confident that by adding an additional distance sensor, we could simply incorporate the distance values into the lookup table and make the automatic correction usable again. Of course, the lookup table could become very large if we do this. There are two avenues around this:

- Store the look-up table in a larger memory, e.g ZBT.
- A more scalable approach is to start doing interpolation in the hardware itself. This will require additional hardware resources and more logic, but the size of the logic will only grow with the number of data points, as opposed to the number of bits coming from the sensors.

Within the audio component, there are many avenues for substantial improvement. The major unresolved issue was the loss of data samples when transferring data over USB and recording it to flash memory. The preferred alternative would be a more modern hardware platform with an updated flash hardware component, as well as an on-board USB module and reference code supplied from the manufacturer. This would likely solve whatever bottleneck stopped us. Another alternative would be to use the serial port instead of the USB hardware, since the labkit already has a serial port built in. This would require a moderately complex Verilog module to implement the serial protocol, although Xilinx may provide some reference code.

Another opportunity for improvement would be variable length tracks. Every track was one second long, which resulted in artificial sounding two digit numbers because some portion of the one second track was silence. A variable length track would reduce the length of silence and sound overall more natural. Our implementation barred us from pursing this improvement as it requires an end-of-track marker for the track. Since we were randomly losing data samples, we could not reliably determine when the track ended, and so were forced into the fixed-length option.

Another useful feature which we unfortunately did not have time to implement is persistent storage of a user's manual correction settings. After all, no matter how good the accel_lut is, it is extremely likely that with certain extreme orientations, the settings obtained through the look-up table are unsatisfactory. One option for dealing with this would be saving manual override settings into some non-volatile storage such as flash, and using them on the next power cycle. Given our issues with compact flash in the audio domain, progress on this front would likely be negligible. Hopefully, with an improved labkit, we would not run into the flash issues.

9 Conclusion

Our projector tilt compensation system used digital logic to successfully perform a perspective transform on an image. This enables fully general manual correction of distortions of the projected image on the screen. Moreover, we also created an automatic correction system for correction of distortion resulting from 2 axes of tilt via the usage of an accelerometer that would sense the angle of tilt along these two axes. We also implemented a useful audio features regarding the percentage of pixels kept by our correction system.

We believe that there were a number of factors that allowed us to succeed with this project. First and foremost, we started early and tackled potential issues as early as possible, with a consistent time commitment every week. Second, we spent a lot of time doing software simulations/verifications before diving in and writing Verilog. The rationale behind this is that correction and testing in software is much faster than synthesis on the FPGA. Third, we had regular meetings among ourselves to ensure that issues some member of our team had would be addressed collectively.

Overall, we are satisfied with the quality of this system, especially considering the time limitations/scope of this project. We are confident that with some additional work, this system could truly rival commercial offerings in this space.

References

Baoxin Li and I. Sezan. Automatic keystone correction for smart projectors with embedded camera. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 4, pages 2829–2832 Vol. 4, Oct 2004. doi: 10.1109/ICIP.2004.1421693.

Ramesh Raskar and Paul Beardsley. A self-correcting projector. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–504. IEEE, 2001.

Rahul Sukthankar, Robert G Stockton, and Matthew D Mullin. Smarter presentations: Exploiting homography in camera-projector systems. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 247–253. IEEE, 2001.

A Source Code

Source code for the project may be obtained on GitHub: `https://github.com/gajjanag/6111_Project`. For completeness, we include all source code here as well. For ease of browsing through the code, we have divided the modules into roughly three categories:

- Staff Modules: modules that are essentially the same as staff provided modules, with minor modifications for our specific needs.
- Labkit: top level labkit module with general instantiations, including clock generators, and ucf file
- Our Modules: modules created by us for our project

A.1 Staff Modules

A.1.1 debounce.v

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce #(parameter DELAY=270000)   // .01 sec with a 27Mhz clock
                (input reset, clock, noisy,
                 output reg clean);

   reg [18:0] count;
   reg new;

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

endmodule
```

A.1.2 delay.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    23:13:26 12/02/2014
// Design Name:
// Module Name:    delay
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
// pulse synchronizer
module synchronize #(parameter NSYNC = 2, parameter W = 1)  // number of sync flops.  must
                   (input clk, input [W-1:0] in,
                    output reg [W-1:0] out);

  reg [(NSYNC-1)*W-1:0] sync;

  always @ (posedge clk)
  begin
    {out,sync} <= {sync[(NSYNC-1)*W-1:0],in};
  end
endmodule
```

A.1.3 display_16hex.v

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
```

```
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 28-Nov-06 CJT: fixed race condition between CE and RS (thanks Javier!)
//
// This verilog module drives the labkit hex dot matrix displays, and puts
// up 16 hexadecimal digits (8 bytes).  These are passed to the module
// through a 64 bit wire ("data"), asynchronously.
//
///////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data;                // 16 hex nibbles to display

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

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   ////////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   ////////////////////////////////////////////////////////////////////////////

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
        if (reset)
          begin
             count = 0;
             clock = 0;
          end
        else if (count == 26)
          begin
             clock = ~clock;
             count = 5'h00;
```

```
         end
       else
         count = count+1;
    end

  always @(posedge clock_27mhz)
    if (reset)
      reset_count <= 100;
    else
      reset_count <= (reset_count==0) ? 0 : reset_count-1;

  assign dreset = (reset_count != 0);

  assign disp_clock = ~clock;

  ////////////////////////////////////////////////////////////////////////////
  //
  // Display State Machine
  //
  ////////////////////////////////////////////////////////////////////////////

  reg [7:0] state;                // FSM state
  reg [9:0] dot_index;                // index to current dot being clocked out
  reg [31:0] control;                // control register
  reg [3:0] char_index;        // index of current character
  reg [39:0] dots;                // dots for a single digit
  reg [3:0] nibble;                // hex nibble of current character

  assign disp_blank = 1'b0; // low <= not blanked

  always @(posedge clock)
    if (dreset)
      begin
         state <= 0;
         dot_index <= 0;
         control <= 32'h7F7F7F7F;
      end
    else
      casex (state)
        8'h00:
          begin
             // Reset displays
             disp_data_out <= 1'b0;
             disp_rs <= 1'b0; // dot register
             disp_ce_b <= 1'b1;
             disp_reset_b <= 1'b0;
```

```
               dot_index <= 0;
               state <= state+1;
            end

          8’h01:
            begin
               // End reset
               disp_reset_b <= 1’b1;
               state <= state+1;
            end

          8’h02:
            begin
               // Initialize dot register (set all dots to zero)
               disp_ce_b <= 1’b0;
               disp_data_out <= 1’b0; // dot_index[0];
               if (dot_index == 639)
                 state <= state+1;
               else
                 dot_index <= dot_index+1;
            end

          8’h03:
            begin
               // Latch dot data
               disp_ce_b <= 1’b1;
               dot_index <= 31;                // re-purpose to init ctrl reg
               disp_rs <= 1’b1; // Select the control register
               state <= state+1;
            end

          8’h04:
            begin
               // Setup the control register
               disp_ce_b <= 1’b0;
               disp_data_out <= control[31];
               control <= {control[30:0], 1’b0};        // shift left
               if (dot_index == 0)
                 state <= state+1;
               else
                 dot_index <= dot_index-1;
            end

          8’h05:
            begin
               // Latch the control register data / dot data
```

```
           disp_ce_b <= 1'b1;
           dot_index <= 39;                // init for single char
           char_index <= 15;                // start with MS char
           state <= state+1;
           disp_rs <= 1'b0;                 // Select the dot register
        end

      8'h06:
        begin
           // Load the user's dot data into the dot reg, char by char
           disp_ce_b <= 1'b0;
           disp_data_out <= dots[dot_index]; // dot data from msb
           if (dot_index == 0)
             if (char_index == 0)
               state <= 5;                        // all done, latch data
             else
             begin
               char_index <= char_index - 1;        // goto next char
               dot_index <= 39;
             end
           else
             dot_index <= dot_index-1;        // else loop thru all dots
        end

    endcase

always @ (data or char_index)
  case (char_index)
    4'h0:                  nibble <= data[3:0];
    4'h1:                  nibble <= data[7:4];
    4'h2:                  nibble <= data[11:8];
    4'h3:                  nibble <= data[15:12];
    4'h4:                  nibble <= data[19:16];
    4'h5:                  nibble <= data[23:20];
    4'h6:                  nibble <= data[27:24];
    4'h7:                  nibble <= data[31:28];
    4'h8:                  nibble <= data[35:32];
    4'h9:                  nibble <= data[39:36];
    4'hA:                  nibble <= data[43:40];
    4'hB:                  nibble <= data[47:44];
    4'hC:                  nibble <= data[51:48];
    4'hD:                  nibble <= data[55:52];
    4'hE:                  nibble <= data[59:56];
    4'hF:                  nibble <= data[63:60];
  endcase

```

```
   always @(nibble)
     case (nibble)
       4’h0: dots <= 40’b00111110_01010001_01001001_01000101_00111110;
       4’h1: dots <= 40’b00000000_01000010_01111111_01000000_00000000;
       4’h2: dots <= 40’b01100010_01010001_01001001_01001001_01000110;
       4’h3: dots <= 40’b00100010_01000001_01001001_01001001_00110110;
       4’h4: dots <= 40’b00011000_00010100_00010010_01111111_00010000;
       4’h5: dots <= 40’b00100111_01000101_01000101_01000101_00111001;
       4’h6: dots <= 40’b00111100_01001010_01001001_01001001_00110000;
       4’h7: dots <= 40’b00000001_01110001_00001001_00000101_00000011;
       4’h8: dots <= 40’b00110110_01001001_01001001_01001001_00110110;
       4’h9: dots <= 40’b00000110_01001001_01001001_00101001_00011110;
       4’hA: dots <= 40’b01111110_00001001_00001001_00001001_01111110;
       4’hB: dots <= 40’b01111111_01001001_01001001_01001001_00110110;
       4’hC: dots <= 40’b00111110_01000001_01000001_01000001_00100010;
       4’hD: dots <= 40’b01111111_01000001_01000001_01000001_00111110;
       4’hE: dots <= 40’b01111111_01001001_01001001_01001001_01000001;
       4’hF: dots <= 40’b01111111_00001001_00001001_00001001_00000001;
     endcase

endmodule
```

A.1.4 vga.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
‘default_nettype none
///////////////////////////////////////////////////////////////////////////////////////////
// vga: Generate XVGA display signals (640 x 480 @ 60Hz)
// essentially a copy of staff xvga module with different timings
// Credits: timings from Jose’s project (Fall 2011),
// general code from staff xvga module (e.g Lab 3 - pong game)
```

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

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

// VGA (640x480) @ 60 Hz
parameter VGA_HBLANKON  =  10'd639;
parameter VGA_HSYNCON   =  10'd655;
parameter VGA_HSYNCOFF  =  10'd751;
parameter VGA_HRESET    =  10'd799;
parameter VGA_VBLANKON  =  10'd479;
parameter VGA_VSYNCON   =  10'd490;
parameter VGA_VSYNCOFF  =  10'd492;
parameter VGA_VRESET    =  10'd523;

// horizontal: 800 pixels total
// display 640 pixels per line
reg hblank,vblank;
wire hsyncon,hsyncoff,hreset,hblankon;
assign hblankon = (hcount == VGA_HBLANKON);
assign hsyncon = (hcount == VGA_HSYNCON);
assign hsyncoff = (hcount == VGA_HSYNCOFF);
assign hreset = (hcount == VGA_HRESET);

// vertical: 524 lines total
// display 480 lines
wire vsyncon,vsyncoff,vreset,vblankon;
assign vblankon = hreset & (vcount == VGA_VBLANKON);
assign vsyncon = hreset & (vcount == VGA_VSYNCON);
assign vsyncoff = hreset & (vcount == VGA_VSYNCOFF);
assign vreset = hreset & (vcount == VGA_VRESET);

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

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

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

A.1.5 divider.v

```
// The divider module divides one number by another. It
// produces a signal named "ready" when the quotient output
// is ready, and takes a signal named "start" to indicate
// the the input dividend and divider is ready.
// sign -- 0 for unsigned, 1 for twos complement

// It uses a simple restoring divide algorithm.
// http://en.wikipedia.org/wiki/Division_(digital)#Restoring_division

module divider #(parameter WIDTH = 8)
  (input clk, sign, start,
   input [WIDTH-1:0] dividend,
   input [WIDTH-1:0] divider,
   output reg [WIDTH-1:0] quotient,
   output [WIDTH-1:0] remainder,
   output ready);

   reg [WIDTH-1:0]  quotient_temp;
   reg [WIDTH*2-1:0] dividend_copy, divider_copy, diff;
   reg negative_output;

   assign remainder = (!negative_output) ?
             dividend_copy[WIDTH-1:0] : ~dividend_copy[WIDTH-1:0] + 1'b1;

   reg [6:0] bit;
   reg del_ready = 1;
   assign ready = (!bit) & ~del_ready;

   wire [WIDTH-2:0] zeros = 0;
   initial bit = 0;
   initial negative_output = 0;
   always @( posedge clk ) begin
      del_ready <= !bit;
      if( start ) begin

         bit = WIDTH;
         quotient = 0;
         quotient_temp = 0;
         dividend_copy = (!sign || !dividend[WIDTH-1]) ?
```

```
                         {1'b0,zeros,dividend} :
                         {1'b0,zeros,~dividend + 1'b1};
         divider_copy = (!sign || !divider[WIDTH-1]) ?
                         {1'b0,divider,zeros} :
                         {1'b0,~divider + 1'b1,zeros};

         negative_output = sign &&
                           ((divider[WIDTH-1] && !dividend[WIDTH-1])
                            ||(!divider[WIDTH-1] && dividend[WIDTH-1]));
       end
      else if ( bit > 0 ) begin
         diff = dividend_copy - divider_copy;
         quotient_temp = quotient_temp << 1;
         if( !diff[WIDTH*2-1] ) begin
            dividend_copy = diff;
            quotient_temp[0] = 1'd1;
         end
         quotient = (!negative_output) ?
                    quotient_temp :
                    ~quotient_temp + 1'b1;
         divider_copy = divider_copy >> 1;
         bit = bit - 1'b1;
      end
   end
endmodule
```

A.1.6 ycrcb2rgb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:05:46 12/02/2014
// Design Name:
// Module Name:    ycrcb2rgb
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
////////////////////////////////////////////////////////////////////////////////////
module ycrcb2rgb ( R, G, B, clk, rst, Y, Cr, Cb );

output [7:0]  R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
      Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
      end
   else
      begin
          Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
      end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     A_int <= (const2 * (Cr_reg - 'd512));
     B1_int <= (const3 * (Cr_reg - 'd512));
     B2_int <= (const4 * (Cb_reg - 'd512));
     C_int <= (const5 * (Cb_reg - 'd512));
     end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     R_int <= X_int + A_int;
     G_int <= X_int - B1_int - B2_int;
     B_int <= X_int + C_int;
     end



/*always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     R_int <= X_int + (const2 * (Cr_reg - 'd512));
     G_int <= X_int - (const3 * (Cr_reg - 'd512)) - (const4 * (Cb_reg - 'd512));
     B_int <= X_int + (const5 * (Cb_reg - 'd512));
     end

*/
/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

A.1.7 ntsc2zbt.v

```
/**
NOTE: Code borrowed heavily from Pranav Kaundinya, et. al. (2012).
*/

module ntsc_to_bram(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we, sw);

   input          clk;        // system clock
   input          vclk;        // video clock from camera
   input [2:0]          fvh;
   input          dv;
   input [29:0]          din;
        input sw;
   output reg [16:0] ntsc_addr;
   output reg [11:0] ntsc_data;
   output          ntsc_we;        // write enable for NTSC data
   parameter          COL_START = 10'd0;
   parameter          ROW_START = 10'd0;

           reg [9:0]          col = 0;
        reg [9:0]          row = 0;
        reg [29:0]          vdata = 0;
        reg                 vwe;
        reg                 old_dv;
        reg                 old_frame;        // frames are even / odd interlaced
        reg                 even_odd;        // decode interlaced frame to this wire

        wire          frame = fvh[2];
        wire          frame_edge = frame & ~old_frame;

        always @ (posedge vclk) begin//LLC1 is reference

                        old_dv <= dv;
                        vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it

                        old_frame <= frame;
                        even_odd = frame_edge ? ~even_odd : even_odd;

                        if (!fvh[2]) begin
                             col <= fvh[0] ? COL_START :
                                    (!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col + 1 : c
                             row <= fvh[1] ? ROW_START :
                                    (!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;
                             vdata <= (dv && !fvh[2]) ? din : vdata;
                          end
        end

   // synchronize with system clock

        reg [9:0] x[1:0],y[1:0];
        reg [29:0] data[1:0];
        reg      we[1:0];
        reg          eo[1:0];

```

```
        always @(posedge clk)begin

                {x[1],x[0]} <= {x[0],col};
                {y[1],y[0]} <= {y[0],row};
                {data[1],data[0]} <= {data[0],vdata};
                {we[1],we[0]} <= {we[0],vwe};
                {eo[1],eo[0]} <= {eo[0],even_odd};
     end

   // edge detection on write enable signal

           reg old_we;
        wire we_edge = we[1] & ~old_we;
        always @(posedge clk) old_we <= we[1];

   // shift each set of four bytes into a large register for the ZBT

           // compute address to store data in
           wire [9:0] y_addr = {y[1][8:0], eo[1]};
           wire [9:0] x_addr = x[1];

                wire [7:0] R, G, B;
                ycrcb2rgb conv( R, G, B, clk, 1'b0, data[1][29:20],
                        data[1][19:10], data[1][9:0] );

           wire [16:0] myaddr_o = (y_addr[7:0] << 8) + (y_addr[7:0] << 6) + x_addr[8:0];
                wire [16:0] myaddr;
                synchronize #(.NSYNC(3), .W(17)) myaddr_sync(clk, myaddr_o, myaddr);
   // update the output address and data only when four bytes ready

           wire ntsc_we_o = (x_addr < COL_START + 10'd320 && y_addr < ROW_START + 10'd240) &

                synchronize #(.NSYNC(3)) we_sync(clk, ntsc_we_o, ntsc_we);
           always @(posedge clk)
             if ( ntsc_we ) begin
                                  ntsc_addr <= myaddr;        // normal and expanded modes
                                  ntsc_data <= ~sw ? {R[7:4], G[7:4], B[7:4]} :
                                        {x_addr[9], 3'b0, x_addr[8], 3'b0, x_addr[7], 3'b0};
               end

endmodule // ntsc_to_zbt
```

A.1.8 video_decoder.v

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

/////////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
   output [29:0] ycrcb;
   output         f;
   output         v;
   output         h;
   output         data_valid;
   // output [4:0] state;

   parameter         SYNC_1 = 0;
   parameter         SYNC_2 = 1;
   parameter         SYNC_3 = 2;
   parameter         SAV_f1_cb0 = 3;
   parameter         SAV_f1_y0 = 4;
   parameter         SAV_f1_cr1 = 5;
```

```
    parameter         SAV_f1_y1 = 6;
    parameter         EAV_f1 = 7;
    parameter         SAV_VBI_f1 = 8;
    parameter         EAV_VBI_f1 = 9;
    parameter         SAV_f2_cb0 = 10;
    parameter         SAV_f2_y0 = 11;
    parameter         SAV_f2_cr1 = 12;
    parameter         SAV_f2_y1 = 13;
    parameter         EAV_f2 = 14;
    parameter         SAV_VBI_f2 = 15;
    parameter         EAV_VBI_f2 = 16;




    // In the start state, the module doesn't know where
    // in the sequence of pixels, it is looking.

    // Once we determine where to start, the FSM goes through a normal
    // sequence of SAV process_YCrCb EAV... repeat

    // The data stream looks as follows
    // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV sequen
    // There are two things we need to do:
    //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
    //   2. Decode the subsequent data

    reg [4:0]         current_state = 5'h00;
    reg [9:0]         y = 10'h000;  // luminance
    reg [9:0]         cr = 10'h000; // chrominance
    reg [9:0]         cb = 10'h000; // more chrominance

    assign         state = current_state;

    always @ (posedge clk)
      begin
         if (reset)
           begin

           end
         else
           begin
              // these states don't do much except allow us to know where we are in the strea
              // whenever the synchronization code is seen, go back to the sync_state before
              // transitioning to the new state
              case (current_state)
```

```
            SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
            SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
            SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                     (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                                     (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                     (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                     (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                     (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                                     (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                     (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

            SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
            SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cr1;
            SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y1;
            SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;

            SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y0;
            SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cr1;
            SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y1;
            SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cb0;

            // These states are here in the event that we want to cover these signals
            // in the future. For now, they just send the state machine back to SYNC_1
            EAV_f1: current_state <= SYNC_1;
            SAV_VBI_f1: current_state <= SYNC_1;
            EAV_VBI_f1: current_state <= SYNC_1;
            EAV_f2: current_state <= SYNC_1;
            SAV_VBI_f2: current_state <= SYNC_1;
            EAV_VBI_f2: current_state <= SYNC_1;

          endcase
       end
  end // always @ (posedge clk)

// implement our decoding mechanism

wire y_enable;
wire cr_enable;
wire cb_enable;

// if y is coming in, enable the register
// likewise for cr and cb
assign y_enable = (current_state == SAV_f1_y0) ||
                  (current_state == SAV_f1_y1) ||
                  (current_state == SAV_f2_y0) ||
                  (current_state == SAV_f2_y1);
```

```
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg           f = 0;

   always @ (posedge clk)
     begin
        y <= y_enable ? tv_in_ycrcb : y;
        cr <= cr_enable ? tv_in_ycrcb : cr;
        cb <= cb_enable ? tv_in_ycrcb : cb;
        f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
     end

endmodule



///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////
// Register 0
///////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                             4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                               4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

///////////////////////////////////////////////////////////////////////////////
// Register 1
///////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                           2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                    1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                      1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM, `FOUR_TIMES_SAMPLING,

///////////////////////////////////////////////////////////////////////////////
// Register 2
///////////////////////////////////////////////////////////////////////////////
```

```

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

///////////////////////////////////////////////////////////////////////////////
// Register 3
///////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT, `INTERFAC

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

```
// Register 4
////////////////////////////////////////////////////////////////////////////////

‘define OUTPUT_DATA_RANGE                       1’b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
‘define BT656_TYPE                              1’b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

‘define ADV7185_REGISTER_4 {‘BT656_TYPE, 3’b000, 3’b110, ‘OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////////


‘define GENERAL_PURPOSE_OUTPUTS                 4’b0000
‘define GPO_0_1_ENABLE                          1’b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
‘define GPO_2_3_ENABLE                          1’b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
‘define BLANK_CHROMA_IN_VBI                     1’b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
‘define HLOCK_ENABLE                            1’b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

‘define ADV7185_REGISTER_5 {‘HLOCK_ENABLE, ‘BLANK_CHROMA_IN_VBI, ‘GPO_2_3_ENABLE, ‘GPO_0_1_

////////////////////////////////////////////////////////////////////////////////
// Register 7
////////////////////////////////////////////////////////////////////////////////

‘define FIFO_FLAG_MARGIN                        5’h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
‘define FIFO_RESET                              1’b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
‘define AUTOMATIC_FIFO_RESET                    1’b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
‘define FIFO_FLAG_SELF_TIME                     1’b1
```

```
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

‘define ADV7185_REGISTER_7 {‘FIFO_FLAG_SELF_TIME, ‘AUTOMATIC_FIFO_RESET, ‘FIFO_RESET, ‘FIFO_

///////////////////////////////////////////////////////////////////////////////
// Register 8
///////////////////////////////////////////////////////////////////////////////

‘define INPUT_CONTRAST_ADJUST                         8’h80

‘define ADV7185_REGISTER_8 {‘INPUT_CONTRAST_ADJUST}

///////////////////////////////////////////////////////////////////////////////
// Register 9
///////////////////////////////////////////////////////////////////////////////

‘define INPUT_SATURATION_ADJUST                       8’h8C

‘define ADV7185_REGISTER_9 {‘INPUT_SATURATION_ADJUST}

///////////////////////////////////////////////////////////////////////////////
// Register A
///////////////////////////////////////////////////////////////////////////////

‘define INPUT_BRIGHTNESS_ADJUST                       8’h00

‘define ADV7185_REGISTER_A {‘INPUT_BRIGHTNESS_ADJUST}

///////////////////////////////////////////////////////////////////////////////
// Register B
///////////////////////////////////////////////////////////////////////////////

‘define INPUT_HUE_ADJUST                              8’h00

‘define ADV7185_REGISTER_B {‘INPUT_HUE_ADJUST}

///////////////////////////////////////////////////////////////////////////////
// Register C
///////////////////////////////////////////////////////////////////////////////

‘define DEFAULT_VALUE_ENABLE                    1’b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
‘define DEFAULT_VALUE_AUTOMATIC_ENABLE          1’b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                        6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFAULT_VALU

//////////////////////////////////////////////////////////////////////////////
// Register D
//////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                       4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                       4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

//////////////////////////////////////////////////////////////////////////////
// Register E
//////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE             1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL            2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE               4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE, `TEMPORAL_DECIMATION_CONTROL,

//////////////////////////////////////////////////////////////////////////////
// Register F
//////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                     2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY             1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
```

```
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP, `POWER_DOWN_LL

///////////////////////////////////////////////////////////////////////////////
// Register 33
///////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE, `AVERAGE_BIRIGHTNESS_LI

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
```

```
‘define ADV7185_REGISTER_14 8’h18
‘define ADV7185_REGISTER_15 8’h60
‘define ADV7185_REGISTER_16 8’h00
‘define ADV7185_REGISTER_17 8’h01
‘define ADV7185_REGISTER_18 8’h00
‘define ADV7185_REGISTER_19 8’h10
‘define ADV7185_REGISTER_1A 8’h10
‘define ADV7185_REGISTER_1B 8’hF0
‘define ADV7185_REGISTER_1C 8’h16
‘define ADV7185_REGISTER_1D 8’h01
‘define ADV7185_REGISTER_1E 8’h00
‘define ADV7185_REGISTER_1F 8’h3D
‘define ADV7185_REGISTER_20 8’hD0
‘define ADV7185_REGISTER_21 8’h09
‘define ADV7185_REGISTER_22 8’h8C
‘define ADV7185_REGISTER_23 8’hE2
‘define ADV7185_REGISTER_24 8’h1F
‘define ADV7185_REGISTER_25 8’h07
‘define ADV7185_REGISTER_26 8’hC2
‘define ADV7185_REGISTER_27 8’h58
‘define ADV7185_REGISTER_28 8’h3C
‘define ADV7185_REGISTER_29 8’h00
‘define ADV7185_REGISTER_2A 8’h00
‘define ADV7185_REGISTER_2B 8’hA0
‘define ADV7185_REGISTER_2C 8’hCE
‘define ADV7185_REGISTER_2D 8’hF0
‘define ADV7185_REGISTER_2E 8’h00
‘define ADV7185_REGISTER_2F 8’hF0
‘define ADV7185_REGISTER_30 8’h00
‘define ADV7185_REGISTER_31 8’h70
‘define ADV7185_REGISTER_32 8’h00
‘define ADV7185_REGISTER_34 8’h0F
‘define ADV7185_REGISTER_35 8’h01
‘define ADV7185_REGISTER_36 8’h00
‘define ADV7185_REGISTER_37 8’h00
‘define ADV7185_REGISTER_38 8’h00
‘define ADV7185_REGISTER_39 8’h00
‘define ADV7185_REGISTER_3A 8’h00
‘define ADV7185_REGISTER_3B 8’h00

‘define ADV7185_REGISTER_44 8’h41
‘define ADV7185_REGISTER_45 8’hBB

‘define ADV7185_REGISTER_F1 8’hEF
‘define ADV7185_REGISTER_F2 8’h80

```

```

module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
      $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
      $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
      $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
      $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
      $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
      $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
      $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
      $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
   end

   //
   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
   //

   reg [7:0] clk_div_count, reset_count;
   reg clock_slow;
   wire reset_slow;

   initial
     begin
        clk_div_count <= 8'h00;
        // synthesis attribute init of clk_div_count is "00";
        clock_slow <= 1'b0;
        // synthesis attribute init of clock_slow is "0";
     end
```

```

always @(posedge clock_27mhz)
  if (clk_div_count == 26)
    begin
       clock_slow <= ~clock_slow;
       clk_div_count <= 0;
    end
  else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
     begin
        state <= 0;
        load <= 0;
        tv_in_reset_b <= 0;
        old_source <= 0;
     end
```

```
        else
          case (state)
            8'h00:
              begin
                 // Assert reset
                 load <= 1'b0;
                 tv_in_reset_b <= 1'b0;
                 if (!ack)
                   state <= state+1;
              end
            8'h01:
              state <= state+1;
            8'h02:
              begin
                 // Release reset
                 tv_in_reset_b <= 1'b1;
                 state <= state+1;
                            end
            8'h03:
              begin
                 // Send ADV7185 address
                 data <= 8'h8A;
                 load <= 1'b1;
                 if (ack)
                   state <= state+1;
              end
            8'h04:
              begin
                 // Send subaddress of first register
                 data <= 8'h00;
                 if (ack)
                   state <= state+1;
              end
            8'h05:
              begin
                 // Write to register 0
                 data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
                 if (ack)
                   state <= state+1;
              end
            8'h06:
              begin
                 // Write to register 1
                 data <= `ADV7185_REGISTER_1;
                 if (ack)
                   state <= state+1;
              end
            8'h07:
              begin
                 // Write to register 2
                 data <= `ADV7185_REGISTER_2;
                 if (ack)
                   state <= state+1;
              end
            8'h08:
              begin
                 // Write to register 3
                 data <= `ADV7185_REGISTER_3;
                 if (ack)
                   state <= state+1;
              end
            8'h09:
              begin
                 // Write to register 4
                 data <= `ADV7185_REGISTER_4;
                 if (ack)
                   state <= state+1;
              end
            8'h0A:
              begin
                 // Write to register 5
                 data <= `ADV7185_REGISTER_5;
                 if (ack)
                   state <= state+1;
              end
            8'h0B:
              begin
                 // Write to register 6
                 data <= 8'h00; // Reserved register, write all zeros
                 if (ack)
                   state <= state+1;
              end
            8'h0C:
              begin
                 // Write to register 7
                 data <= `ADV7185_REGISTER_7;
                 if (ack)
                   state <= state+1;
              end
            8'h0D:
              begin
                 // Write to register 8
                 data <= `ADV7185_REGISTER_8;
                 if (ack)
                   state <= state+1;
              end
            8'h0E:
              begin
                 // Write to register 9
                 data <= `ADV7185_REGISTER_9;
                 if (ack)
                   state <= state+1;
              end
            8'h0F: begin
               // Write to register A
               data <= `ADV7185_REGISTER_A;
             if (ack)
               state <= state+1;
            end
            8'h10:
              begin
                 // Write to register B
                 data <= `ADV7185_REGISTER_B;
                 if (ack)
                   state <= state+1;
              end
            8'h11:
              begin
                 // Write to register C
                 data <= `ADV7185_REGISTER_C;
                 if (ack)
                   state <= state+1;
              end
            8'h12:
              begin
                 // Write to register D
                 data <= `ADV7185_REGISTER_D;
                 if (ack)
                   state <= state+1;
              end
            8'h13:
              begin
                 // Write to register E
                 data <= `ADV7185_REGISTER_E;
                 if (ack)
                   state <= state+1;
              end
            8'h14:
              begin
                 // Write to register F
                 data <= ‘ADV7185_REGISTER_F;
                 if (ack)
                   state <= state+1;
              end
            8’h15:
              begin
                 // Wait for I2C transmitter to finish
                 load <= 1’b0;
                 if (idle)
                   state <= state+1;
              end
            8’h16:
              begin
                 // Write address
                 data <= 8’h8A;
                 load <= 1’b1;
                 if (ack)
                   state <= state+1;
              end
            8’h17:
              begin
                 data <= 8’h33;
                 if (ack)
                   state <= state+1;
              end
            8’h18:
              begin
                 data <= ‘ADV7185_REGISTER_33;
                 if (ack)
                   state <= state+1;
              end
            8’h19:
              begin
                 load <= 1’b0;
                 if (idle)
                   state <= state+1;
              end

            8’h1A: begin
               data <= 8’h8A;
               load <= 1’b1;
               if (ack)
                 state <= state+1;
            end
            8'h1B:
              begin
                 data <= 8'h33;
                 if (ack)
                   state <= state+1;
              end
            8'h1C:
              begin
                 load <= 1'b0;
                 if (idle)
                   state <= state+1;
              end
            8'h1D:
              begin
                 load <= 1'b1;
                 data <= 8'h8B;
                 if (ack)
                   state <= state+1;
              end
            8'h1E:
              begin
                 data <= 8'hFF;
                 if (ack)
                   state <= state+1;
              end
            8'h1F:
              begin
                 load <= 1'b0;
                 if (idle)
                   state <= state+1;
              end
            8'h20:
              begin
                 // Idle
                 if (old_source != source) state <= state+1;
                 old_source <= source;
              end
            8'h21: begin
               // Send ADV7185 address
               data <= 8'h8A;
               load <= 1'b1;
               if (ack) state <= state+1;
            end
            8'h22: begin
               // Send subaddress of register 0
               data <= 8'h00;
```

```
             if (ack) state <= state+1;
          end
          8’h23: begin
             // Write to register 0
             data <= ‘ADV7185_REGISTER_0 | {5’h00, {3{source}}};
             if (ack) state <= state+1;
          end
          8’h24: begin
             // Wait for I2C transmitter to finish
             load <= 1’b0;
             if (idle) state <= 8’h20;
          end
       endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;

   assign sda = sdai ? 1’bZ : 1’b0;

   always @(posedge clock4x)
     if (reset)
       begin
          state <= 0;
          ack <= 0;
       end
     else
       case (state)
```

```
            8'h00: // idle
              begin
                 scl <= 1'b1;
                 sdai <= 1'b1;
                 ack <= 1'b0;
                 idle <= 1'b1;
                 if (load)
                   begin
                      ldata <= data;
                      ack <= 1'b1;
                      state <= state+1;
                   end
              end
            8'h01: // Start
              begin
                 ack <= 1'b0;
                 idle <= 1'b0;
                 sdai <= 1'b0;
                 state <= state+1;
              end
            8'h02:
              begin
                 scl <= 1'b0;
                 state <= state+1;
              end
            8'h03: // Send bit 7
              begin
                 ack <= 1'b0;
                 sdai <= ldata[7];
                 state <= state+1;
              end
            8'h04:
              begin
                 scl <= 1'b1;
                 state <= state+1;
              end
            8'h05:
              begin
                 state <= state+1;
              end
            8'h06:
              begin
                 scl <= 1'b0;
                 state <= state+1;
              end
            8'h07:
```

```
            begin
               sdai <= ldata[6];
               state <= state+1;
            end
          8'h08:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h09:
            begin
               state <= state+1;
            end
          8'h0A:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h0B:
            begin
               sdai <= ldata[5];
               state <= state+1;
            end
          8'h0C:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h0D:
            begin
               state <= state+1;
            end
          8'h0E:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h0F:
            begin
               sdai <= ldata[4];
               state <= state+1;
            end
          8'h10:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8’h11:
            begin
               state <= state+1;
            end
          8’h12:
            begin
               scl <= 1’b0;
               state <= state+1;
            end
          8’h13:
            begin
               sdai <= ldata[3];
               state <= state+1;
            end
          8’h14:
            begin
               scl <= 1’b1;
               state <= state+1;
            end
          8’h15:
            begin
               state <= state+1;
            end
          8’h16:
            begin
               scl <= 1’b0;
               state <= state+1;
            end
          8’h17:
            begin
               sdai <= ldata[2];
               state <= state+1;
            end
          8’h18:
            begin
               scl <= 1’b1;
               state <= state+1;
            end
          8’h19:
            begin
               state <= state+1;
            end
          8’h1A:
            begin
               scl <= 1’b0;
               state <= state+1;
            end
          8'h1B:
            begin
               sdai <= ldata[1];
               state <= state+1;
            end
          8'h1C:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h1D:
            begin
               state <= state+1;
            end
          8'h1E:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h1F:
            begin
               sdai <= ldata[0];
               state <= state+1;
            end
          8'h20:
            begin
               scl <= 1'b1;
               state <= state+1;
            end
          8'h21:
            begin
               state <= state+1;
            end
          8'h22:
            begin
               scl <= 1'b0;
               state <= state+1;
            end
          8'h23: // Acknowledge bit
            begin
               state <= state+1;
            end
          8'h24:
            begin
```

```
                scl <= 1'b1;
                state <= state+1;
             end
           8'h25:
             begin
                state <= state+1;
             end
           8'h26:
             begin
                scl <= 1'b0;
                if (load)
                  begin
                     ldata <= data;
                     ack <= 1'b1;
                     state <= 3;
                  end
                else
                  state <= state+1;
             end
           8'h27:
             begin
                sdai <= 1'b0;
                state <= state+1;
             end
           8'h28:
             begin
                scl <= 1'b1;
                state <= state+1;
             end
           8'h29:
             begin
                sdai <= 1'b1;
                state <= 0;
             end
       endcase

endmodule


```

A.1.9 flash_int.v

```
//flash interface
module flash_int(reset, clock, op, address, wdata, rdata, busy, flash_data,
                 flash_address, flash_ce_b, flash_oe_b, flash_we_b,
                 flash_reset_b, flash_sts, flash_byte_b);
```

```

  parameter access_cycles = 5;
  parameter reset_assert_cycles = 1000;
  parameter reset_recovery_cycles = 30;

  input reset, clock; // Reset and clock for the flash interface
  input [1:0] op; // Flash operation select (read, write, idle)
  input [22:0] address;
  input [15:0] wdata;
  output [15:0] rdata;
  output busy;
  inout [15:0] flash_data;
  output [23:0] flash_address;
  output flash_ce_b, flash_oe_b, flash_we_b;
  output flash_reset_b, flash_byte_b;
  input  flash_sts;

  reg [1:0] lop;
  reg [15:0] rdata;
  reg busy;
  reg [15:0] flash_wdata;
  reg flash_ddata;
  reg [23:0] flash_address;
  reg flash_oe_b, flash_we_b, flash_reset_b;

  assign flash_ce_b = flash_oe_b && flash_we_b;
  assign flash_byte_b = 1; // 1 = 16-bit mode (A0 ignored)

  assign flash_data = flash_ddata ? flash_wdata : 16'hZ;

  initial
    flash_reset_b <= 1'b1;

  reg [9:0] state;

  always @(posedge clock)
    if (reset)
      begin
                       state <= 0;
                       flash_reset_b <= 0;
                       flash_we_b <= 1;
                       flash_oe_b <= 1;
                       flash_ddata <= 0;
                       busy <= 1;
      end
    else if (flash_reset_b == 0)
```

```
      if (state == reset_assert_cycles)
                      begin
                              flash_reset_b <= 1;
                              state <= 1023-reset_recovery_cycles;
                      end
      else
                      state <= state+1;
        else if ((state == 0) && !busy)
      // The flash chip and this state machine are both idle. Latch the user's
      // address and write data inputs. Deassert OE and WE, and stop driving
      // the data buss ourselves. If a flash operation (read or write) is
      // requested, move to the next state.
      begin
                      flash_address <= {address, 1'b0};
                      flash_we_b <= 1;
                      flash_oe_b <= 1;
                      flash_ddata <= 0;
                      flash_wdata <= wdata;
                      lop <= op;
                      if (op != `FLASHOP_IDLE)
                              begin
                                      busy <= 1;
                                      state <= state+1;
                              end
                      else
                              busy <= 0;
                      end
               else if ((state==0) && flash_sts)
                      busy <= 0;
               else if (state == 1)
      // The first stage of a flash operation. The address bus is already set,
      // so, if this is a read, we assert OE. For a write, we start driving
      // the user's data onto the flash databus (the value was latched in the
      // previous state.
                      begin
                              if (lop == `FLASHOP_WRITE)
                                      flash_ddata <= 1;
                              else if (lop == `FLASHOP_READ)
                                      flash_oe_b <= 0;
                              state <= state+1;
      end
    else if (state == 2)
      // The second stage of a flash operation. Nothing to do for a read. For
      // a write, we assert WE.
      begin
                      if (lop == `FLASHOP_WRITE)
```

```
                                flash_we_b <= 0;
                        state <= state+1;
       end
     else if (state == access_cycles+1)
       // The third stage of a flash operation. For a read, we latch the data
       // from the flash chip. For a write, we deassert WE.
       begin
                        if (lop == `FLASHOP_WRITE)
                                flash_we_b <= 1;
                        if (lop == `FLASHOP_READ)
                                rdata <= flash_data;
                        state <= 0;
       end
     else
       begin
                        if (!flash_sts)
                                busy <= 1;
                        state <= state+1;
       end

endmodule
```

A.1.10 flash_manager.v

```
//manages all the stuff needed to read and write to the flash ROM
module flash_manager(
        clock, reset,
        dots,
        writemode,
        wdata,
        dowrite,
        raddr,
        frdata,
        doread,
        busy,
        flash_data,
        flash_address,
        flash_ce_b,
        flash_oe_b,
        flash_we_b,
        flash_reset_b,
        flash_sts,
        flash_byte_b,
        fsmstate);

        input reset, clock;                        //clock and reset
        output [639:0] dots;                //outputs to dot-matrix to help debug flash, no
        input writemode;                        //if true then we're in write mode, else we
        input [15:0] wdata;                        //data to be written
        input dowrite;                                //putting this high tells the manager
        input [22:0] raddr;                        //address to read from
        output[15:0] frdata;                //data being read
        reg[15:0]    rdata;
        input doread;                                //putting this high tells the manager t
        output busy;                                //and an output to tell folks we're sti
        reg busy;

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

        wire flash_busy;                //except these, which are internal to the interface
        wire[15:0] fwdata;
        wire[15:0] frdata;
        wire[22:0] address;
        wire [1:0] op;

        reg [1:0] mode;
        wire fsm_busy;

        reg[2:0] state;                                        //210

        output[11:0] fsmstate;
        wire [7:0] fsmstateinv;
        assign fsmstate = {state,flash_busy,fsm_busy,fsmstateinv[4:0],mode};        //for d

                                                                                //this guy t
        flash_int flash(reset, clock, op, address, fwdata, frdata, flash_busy, flash_data, f
                                                                                //and this g
        test_fsm  fsm  (reset, clock, op, address, fwdata, frdata, flash_busy, dots, mode, f

        parameter MODE_IDLE        = 0;
        parameter MODE_INIT        = 1;
        parameter MODE_WRITE = 2;
        parameter MODE_READ        = 3;

        parameter HOME                 = 3'd0;
        parameter MEM_INIT         = 3'd1;
        parameter MEM_WAIT         = 3'd2;
        parameter WRITE_READY= 3'd3;
        parameter WRITE_WAIT        = 3'd4;
        parameter READ_READY        = 3'd5;
        parameter READ_WAIT         = 3'd6;

        always @ (posedge clock)
                if(reset)
                        begin
                                busy <= 1;
                                state <= HOME;
                                mode <= MODE_IDLE;
                        end
                else begin
                        case(state)
                                HOME://0                                //we always start h
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        if(writemode)
                                                                begin
                                                                        busy <= 1;
                                                                        state <= MEM_INIT;
                                                                end
                                                        else
                                                                begin
                                                                        busy <= 1;
                                                                        state <= READ_READY;
                                                                end
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                MEM_INIT://1
                                        begin
                                                busy <= 1;
                                                mode <= MODE_INIT;
                                                if(fsm_busy)
                                                        state <= MEM_WAIT;
                                        end

                                MEM_WAIT://2
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        state<= WRITE_READY;
                                                end
                                        else
```

```
                                                mode <= MODE_IDLE;

                                WRITE_READY://3                                         //wa
                                        if(dowrite)
                                                begin
                                                        busy <= 1;
                                                        mode <= MODE_WRITE;
                                                end
                                        else if(busy)
                                                state <= WRITE_WAIT;
                                        else if(!writemode)
                                                state <= READ_READY;

                                WRITE_WAIT://4                                 //waiting for

                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        state <= WRITE_READY;
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                READ_READY://5                                 //ready to re
                                        if(doread)
                                                begin
                                                        busy <= 1;
                                                        mode <= MODE_READ;
                                                        if(busy)                        //l
                                                                state <= READ_WAIT;
                                                end
                                        else
                                                busy <= 0;

                                READ_WAIT://6                         //waiting for flash to
                                        if(!fsm_busy)
                                                begin
                                                        busy <= 0;
                                                        state <= READ_READY;
                                                end
                                        else
                                                mode <= MODE_IDLE;

                                default: begin                //should never happen...
                                        state <= 3'd7;
                                end
                        endcase
```

```
        end
endmodule
```

A.1.11 test_fsm.v

```
`define STATUS_RESET             4'h0
`define STATUS_READ_ID           4'h1
`define STATUS_CLEAR_LOCKS       4'h2
`define STATUS_ERASING           4'h3
`define STATUS_WRITING           4'h4
`define STATUS_READING           4'h5
`define STATUS_SUCCESS           4'h6
`define STATUS_BAD_MANUFACTURER  4'h7
`define STATUS_BAD_SIZE          4'h8
`define STATUS_LOCK_BIT_ERROR    4'h9
`define STATUS_ERASE_BLOCK_ERROR 4'hA
`define STATUS_WRITE_ERROR       4'hB
`define STATUS_READ_WRONG_DATA   4'hC

`define NUM_BLOCKS 128
`define BLOCK_SIZE 64*1024
`define LAST_BLOCK_ADDRESS ((`NUM_BLOCKS-1)*`BLOCK_SIZE)
`define LAST_ADDRESS (`NUM_BLOCKS*`BLOCK_SIZE-1)

`define FLASHOP_IDLE  2'b00
`define FLASHOP_READ  2'b01
`define FLASHOP_WRITE 2'b10

module test_fsm (reset, clock, fop, faddress, fwdata, frdata, fbusy, dots, mode, busy, datai
   input reset, clock;
   output [1:0] fop;
   output [22:0] faddress;
   output [15:0] fwdata;
   input [15:0]  frdata;
   input fbusy;
   output [639:0] dots;
   input [1:0] mode;
   output busy;
   input [15:0] datain;
   input [22:0] addrin;
   output state;

   reg [1:0] fop;
   reg [22:0] faddress;
   reg [15:0] fwdata;
   reg [639:0] dots;
   reg busy;
   reg [15:0] data_to_store;

   ////////////////////////////////////////////////////////////////////////////
   //
   // State Machine
   //
   ////////////////////////////////////////////////////////////////////////////

   reg [7:0] state;
   reg [3:0] status;

   parameter MODE_IDLE        = 0;
   parameter MODE_INIT        = 1;
   parameter MODE_WRITE = 2;
   parameter MODE_READ        = 3;

   parameter MAX_ADDRESS = 23’h200000;

   parameter HOME = 8’h12;


   always @(posedge clock)
     if (reset)
       begin
                        state <= HOME;
                        status <= ‘STATUS_RESET;
                        faddress <= 0;
                        fop <= ‘FLASHOP_IDLE;
                        busy <= 1;
       end
     else if (!fbusy && (fop == ‘FLASHOP_IDLE))
       case (state)

         HOME://12
                case(mode)
                        MODE_INIT: begin
                                state <= 8’h00;
                                busy <= 1;
                        end

                        MODE_WRITE: begin
                                state <= 8’h0C;
                                busy <= 1;
                        end

```

```
                       MODE_READ: begin
                               busy <= 1;
                               if(status == `STATUS_READING)
                                       state <= 8'h11;
                               else
                                       state <= 8'h10;
                       end

                       default: begin
                               state <= HOME;
                               busy <= 0;
                       end
               endcase

        ///////////////////////////////////////////////////////////////////////
        // Wipe It
        ///////////////////////////////////////////////////////////////////////
        8'h00:
               begin
                       // Issue "read id codes" command
                       status <= `STATUS_READ_ID;
                       faddress <= 0;
                       fwdata <= 16'h0090;
                       fop <= `FLASHOP_WRITE;
                       state <= state+1;
               end

        8'h01:
          begin
             // Read manufacturer code
             faddress <= 0;
             fop <= `FLASHOP_READ;
             state <= state+1;
          end

        8'h02:
          if (frdata != 16'h0089) // 16'h0089 = Intel
            status <= `STATUS_BAD_MANUFACTURER;
          else
            begin
               // Read the device size code
               faddress <= 1;
               fop <= `FLASHOP_READ;
               state <= state+1;
            end

        8'h03:
          if (frdata != 16'h0018) // 16'h0018 = 128Mbit
            status <= `STATUS_BAD_SIZE;
          else
            begin
               faddress <= 0;
               fwdata <= 16'hFF;
               fop <= `FLASHOP_WRITE;
               state <= state+1;
            end

        8'h04:
          begin
             // Issue "clear lock bits" command
             status <= `STATUS_CLEAR_LOCKS;
             faddress <= 0;
             fwdata <= 16'h60;
             fop <= `FLASHOP_WRITE;
             state <= state+1;
          end

        8'h05:
          begin
             // Issue "confirm clear lock bits" command
             faddress <= 0;
             fwdata <= 16'hD0;
             fop <= `FLASHOP_WRITE;
             state <= state+1;
          end

        8'h06:
          begin
             // Read status
             faddress <= 0;
             fop <= `FLASHOP_READ;
             state <= state+1;
          end

        8'h07:
          if (frdata[7] == 1) // Done clearing lock bits
            if (frdata[6:1] == 0) // No errors
              begin
                 faddress <= 0;
                 fop <= `FLASHOP_IDLE;
                 state <= state+1;
              end
```

```
    else
      status <= ‘STATUS_LOCK_BIT_ERROR;
  else // Still busy, reread status register
    begin
       faddress <= 0;
       fop <= ‘FLASHOP_READ;
    end

//////////////////////////////////////////////////////////////////////
// Block Erase Sequence
//////////////////////////////////////////////////////////////////////
8’h08:
  begin
     status <= ‘STATUS_ERASING;
     fwdata <= 16’h20; // Issue "erase block" command
     fop <= ‘FLASHOP_WRITE;
     state <= state+1;
  end

8’h09:
  begin
     fwdata <= 16’hD0; // Issue "confirm erase" command
     fop <= ‘FLASHOP_WRITE;
     state <= state+1;
  end
8’h0A:
  begin
     fop <= ‘FLASHOP_READ;
     state <= state+1;
  end
8’h0B:
  if (frdata[7] == 1) // Done erasing block
    if (frdata[6:1] == 0) // No errors
      if (faddress != MAX_ADDRESS) // ‘LAST_BLOCK_ADDRESS)
        begin
           faddress <= faddress+‘BLOCK_SIZE;
           fop <= ‘FLASHOP_IDLE;
           state <= state-3;
        end
      else
        begin
           faddress <= 0;
           fop <= ‘FLASHOP_IDLE;
           state <= HOME;                //done erasing, go home
        end
    else // Erase error detected
```

```
      status <= ‘STATUS_ERASE_BLOCK_ERROR;
  else // Still busy
    fop <= ‘FLASHOP_READ;

//////////////////////////////////////////////////////////////////////
// Write Mode
//////////////////////////////////////////////////////////////////////
8’h0C:
       begin
               data_to_store <= datain;
               status <= ‘STATUS_WRITING;
               fwdata <= 16’h40; // Issue "setup write" command
               fop <= ‘FLASHOP_WRITE;
               state <= state+1;
       end

8’h0D:
  begin
     fwdata <= data_to_store; // Finish write
     fop <= ‘FLASHOP_WRITE;
     state <= state+1;
  end
8’h0E:
  begin
     // Read status register
     fop <= ‘FLASHOP_READ;
     state <= state+1;
  end
8’h0F:
  if (frdata[7] == 1) // Done writing
    if (frdata[6:1] == 0) // No errors
      if (faddress != 23’h7FFFFF) // ‘LAST_ADDRESS)
                       begin
                               faddress <= faddress+1;
                               fop <= ‘FLASHOP_IDLE;
                               state <= HOME;
                       end
                else
                       status <= ‘STATUS_WRITE_ERROR;
    else // Write error detected
      status <= ‘STATUS_WRITE_ERROR;
  else // Still busy
    fop <= ‘FLASHOP_READ;

//////////////////////////////////////////////////////////////////////
// Read Mode INIT
```

```
        ////////////////////////////////////////////////////////////////////////
        8'h10:
          begin
             status <= `STATUS_READING;
             fwdata <= 16'hFF; // Issue "read array" command
             fop <= `FLASHOP_WRITE;
             faddress <= 0;
             state <= state+1;
          end

        ////////////////////////////////////////////////////////////////////////
        // Read Mode
        ////////////////////////////////////////////////////////////////////////
        8'h11:
          begin
                 faddress <= addrin;
             fop <= `FLASHOP_READ;
             state <= HOME;
          end

       default:
               begin
                       status <= `STATUS_BAD_MANUFACTURER;
                       faddress <= 0;
                       state <= HOME;
               end

       endcase
    else
      fop <= `FLASHOP_IDLE;

  function [39:0] nib2char;
     input [3:0] nib;
     begin
        case (nib)
          4'h0: nib2char = 40'b00111110_01010001_01001001_01000101_00111110;
          4'h1: nib2char = 40'b00000000_01000010_01111111_01000000_00000000;
          4'h2: nib2char = 40'b01100010_01010001_01001001_01001001_01000110;
          4'h3: nib2char = 40'b00100010_01000001_01001001_01001001_00110110;
          4'h4: nib2char = 40'b00011000_00010100_00010010_01111111_00010000;
          4'h5: nib2char = 40'b00100111_01000101_01000101_01000101_00111001;
          4'h6: nib2char = 40'b00111100_01001010_01001001_01001001_00110000;
          4'h7: nib2char = 40'b00000001_01110001_00001001_00000101_00000011;
          4'h8: nib2char = 40'b00110110_01001001_01001001_01001001_00110110;
          4'h9: nib2char = 40'b00000110_01001001_01001001_00101001_00011110;
          4'hA: nib2char = 40'b01111110_00001001_00001001_00001001_01111110;
          4'hB: nib2char = 40'b01111111_01001001_01001001_01001001_00110110;
          4'hC: nib2char = 40'b00111110_01000001_01000001_01000001_00100010;
          4'hD: nib2char = 40'b01111111_01000001_01000001_01000001_00111110;
          4'hE: nib2char = 40'b01111111_01001001_01001001_01001001_01000001;
          4'hF: nib2char = 40'b01111111_00001001_00001001_00001001_00000001;
        endcase
     end
  endfunction

  wire [159:0] data_dots;
  assign data_dots = {nib2char(frdata[15:12]), nib2char(frdata[11:8]),
                      nib2char(frdata[7:4]), nib2char(frdata[3:0])};

  wire [239:0] address_dots;
  assign address_dots = {nib2char({ 1'b0, faddress[22:20]}),
                         nib2char(faddress[19:16]),
                         nib2char(faddress[15:12]),
                         nib2char(faddress[11:8]),
                         nib2char(faddress[7:4]),
                         nib2char(faddress[3:0])};

  always @(status or address_dots or data_dots)
    case (status)
      `STATUS_RESET:
        dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                 40'b01111111_01001001_01001001_01001001_01000001, // E
                 40'b00100110_01001001_01001001_01001001_00110010, // S
                 40'b01111111_01001001_01001001_01001001_01000001, // E
                 40'b00000001_00000001_01111111_00000001_00000001, // T
                 40'b00000000_00000000_00000000_00000000_00000000, //
                 40'b00000000_00000000_00000000_00000000_00000000, //
                 40'b00000000_00000000_00000000_00000000_00000000, //
                 40'b00000000_00000000_00000000_00000000_00000000, //
                 40'b00000000_00000000_00000000_00000000_00000000, //
                 40'b00001000_00001000_00001000_00001000_00001000, // -
                 40'b00001000_00001000_00001000_00001000_00001000, // -
                 40'b00001000_00001000_00001000_00001000_00001000, // -
                 40'b00001000_00001000_00001000_00001000_00001000, // -
                 40'b00001000_00001000_00001000_00001000_00001000, // -
                 40'b00001000_00001000_00001000_00001000_00001000};// -
      `STATUS_READ_ID:
        dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                 40'b01111111_01001001_01001001_01001001_01000001, // E
                 40'b01111110_00001001_00001001_00001001_01111110, // A
                 40'b01111111_01000001_01000001_01000001_00111110, // D
                 40'b00000000_00000000_00000000_00000000_00000000, //
```

```
                     40'b00000000_01000001_01111111_01000001_00000000, // I
                     40'b01111111_01000001_01000001_01000001_00111110, // D
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     address_dots};
          `STATUS_CLEAR_LOCKS:
            dots <= {40'b00111110_01000001_01000001_01000001_00100010, // C
                     40'b01111111_01000000_01000000_01000000_01000000, // L
                     40'b01111111_00001001_00011001_00101001_01000110, // R
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b01111111_01000000_01000000_01000000_01000000, // L
                     40'b00111110_01000001_01000001_01000001_00111110, // O
                     40'b00111110_01000001_01000001_01000001_00100010, // C
                     40'b01111111_00001000_00010100_00100010_01000001, // K
                     40'b00100110_01001001_01001001_01001001_00110010, // S
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     address_dots};
          `STATUS_ERASING:
            dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                     40'b01111111_00001001_00011001_00101001_01000110, // R
                     40'b01111110_00001001_00001001_00001001_01111110, // A
                     40'b00100110_01001001_01001001_01001001_00110010, // S
                     40'b00000000_01000001_01111111_01000001_00000000, // I
                     40'b01111111_00000010_00000100_00001000_01111111, // N
                     40'b00111110_01000001_01001001_01001001_00111010, // G
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     address_dots};
          `STATUS_WRITING:
            dots <= {40'b01111111_00100000_00011000_00100000_01111111, // W
                     40'b01111111_00001001_00011001_00101001_01000110, // R
                     40'b00000000_01000001_01111111_01000001_00000000, // I
                     40'b00000001_00000001_01111111_00000001_00000001, // T
                     40'b00000000_01000001_01111111_01000001_00000000, // I
                     40'b01111111_00000010_00000100_00001000_01111111, // N
                     40'b00111110_01000001_01001001_01001001_00111010, // G
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     40'b00000000_00000000_00000000_00000000_00000000, //
                     address_dots};
          `STATUS_READING:
            dots <= {40'b01111111_00001001_00011001_00101001_01000110, // R
                     40'b01111111_01001001_01001001_01001001_01000001, // E
                     40'b01111110_00001001_00001001_00001001_01111110, // A
```

```
                   40'b01111111_01000001_01000001_01000001_00111110, // D
                   40'b00000000_01000001_01111111_01000001_00000000, // I
                   40'b01111111_00000010_00000100_00001000_01111111, // N
                   40'b00111110_01000001_01001001_01001001_00111010, // G
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   address_dots};
        `STATUS_SUCCESS:
          dots <= {40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b01111111_00001001_00001001_00001001_00000110, // P
                   40'b01111110_00001001_00001001_00001001_01111110, // A
                   40'b00100110_01001001_01001001_01001001_00110010, // S
                   40'b00100110_01001001_01001001_01001001_00110010, // S
                   40'b01111111_01001001_01001001_01001001_01000001, // E
                   40'b01111111_01000001_01000001_01000001_00111110, // D
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00101010_00011100_01111111_00011100_00101010, // *
                   40'b00000000_00000000_00000000_00000000_00000000};//
        `STATUS_BAD_MANUFACTURER:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                   40'b01111111_00001001_00011001_00101001_01000110, // R
                   40'b01111111_00001001_00011001_00101001_01000110, // R
                   40'b00000000_00110110_00110110_00000000_00000000, // :
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b01111111_00000010_00001100_00000010_01111111, // M
                   40'b01111110_00001001_00001001_00001001_01111110, // A
                   40'b01111111_00000010_00000100_00001000_01111111, // N
                   40'b01111111_00001001_00001001_00001001_00000001, // U
                   40'b01111111_00001001_00001001_00001001_00000001, // F
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   data_dots};
        `STATUS_BAD_SIZE:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                   40'b01111111_00001001_00011001_00101001_01000110, // R
                   40'b01111111_00001001_00011001_00101001_01000110, // R
                   40'b00000000_00110110_00110110_00000000_00000000, // :
                   40'b00000000_00000000_00000000_00000000_00000000, //
                   40'b00100110_01001001_01001001_01001001_00110010, // S
```

```
                    40'b00000000_01000001_01111111_01000001_00000000, // I
                    40'b01100001_01010001_01001001_01000101_01000011, // Z
                    40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b00000000_00000000_00000000_00000000_00000000,
                    40'b00000000_00000000_00000000_00000000_00000000,
                    40'b00000000_00000000_00000000_00000000_00000000,
                    data_dots};
        `STATUS_LOCK_BIT_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b00000000_00110110_00110110_00000000_00000000, // :
                    40'b00000000_00000000_00000000_00000000_00000000, //
                    40'b01111111_01000000_01000000_01000000_01000000, // L
                    40'b00111110_01000001_01000001_01000001_00111110, // O
                    40'b00111110_01000001_01000001_01000001_00100010, // C
                    40'b01111111_00001000_00010100_00100010_01000001, // K
                    40'b00100110_01001001_01001001_01001001_00110010, // S
                    40'b00000000_00000000_00000000_00000000_00000000,
                    40'b00000000_00000000_00000000_00000000_00000000,
                    data_dots};
        `STATUS_ERASE_BLOCK_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b00000000_00110110_00110110_00000000_00000000, // :
                    40'b00000000_00000000_00000000_00000000_00000000, //
                    40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b01111110_00001001_00001001_00001001_01111110, // A
                    40'b00100110_01001001_01001001_01001001_00110010, // S
                    40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b00000000_00000000_00000000_00000000_00000000,
                    40'b00000000_00000000_00000000_00000000_00000000,
                    data_dots};
        `STATUS_WRITE_ERROR:
          dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b00000000_00110110_00110110_00000000_00000000, // :
                    40'b00000000_00000000_00000000_00000000_00000000, //
                    40'b01111111_00100000_00011000_00100000_01111111, // W
                    40'b01111111_00001001_00011001_00101001_01000110, // R
                    40'b00000000_01000001_01111111_01000001_00000000, // I
                    40'b00000001_00000001_01111111_00000001_00000001, // T
                    40'b01111111_01001001_01001001_01001001_01000001, // E
```

```
                  40'b00000000_00000000_00000000_00000000_00000000,
                  40'b00000000_00000000_00000000_00000000_00000000,
                  data_dots};
       `STATUS_READ_WRONG_DATA:
         dots <= {40'b01111111_01001001_01001001_01001001_01000001, // E
                  40'b01111111_00001001_00011001_00101001_01000110, // R
                  40'b01111111_00001001_00011001_00101001_01000110, // R
                  40'b00000000_00110110_00110110_00000000_00000000, // :
                  40'b00000000_00000000_00000000_00000000_00000000,
                  address_dots,
                  40'b00000000_00000000_00000000_00000000_00000000,
                  data_dots};
         default:
         dots <= {16{40'b01010101_00101010_01010101_00101010_01010101}};
     endcase

endmodule
```

A.1.12 usb_input.v

```
//reads data and puts it on out
module usb_input(clk,reset,data,rd,rxf,out,newout,hold,state);
        input clk, reset;        //clock and reset
        input [7:0] data;        //the data pins from the USB fifo
        input rxf;                         //the rxf pin from the USB fifo
        output rd;                         //the rd pin from the USB fifo
        reg rd;

        output[7:0] out;        //this is where data goes when it has been read from the fi
        reg[7:0] out;
        output newout;                //when this is high, out contains a new chunk of data
        reg newout;
        input hold;                         //as long as hold is high, this module sits
                                                 //still module and will not accept new data

        output state;                //for debugging purposes
        reg[3:0] state;

        parameter RESET                 = 0;                  //state data
        parameter WAIT                        = 1;
        parameter WAIT2                        = 2;
        parameter WAIT3                        = 3;
        parameter DATA_COMING        = 4;
        parameter DATA_COMING_2        = 5;
        parameter DATA_COMING_3        = 6;
        parameter DATA_COMING_4        = 7;
        parameter DATA_COMING_5        = 8;
        parameter DATA_HERE          = 9;
        parameter DATA_LEAVING        =10;
        parameter DATA_LEAVING_2=11;
        parameter DATA_LEAVING_3=12;
        parameter DATA_LEAVING_4=13;
        parameter DATA_LEAVING_5=14;
        parameter DATA_LEAVING_6=15;

        initial
                state <= WAIT;

        always @ (posedge clk)
                if(reset)
                        begin
                                newout <= 0;
                                rd <= 1;                        //we can't read data
                                state <= WAIT;
                        end
                else
                        if(~hold)
                                begin
                                        newout <= 0;
                                        case(state)
                                        WAIT:
                                                if(~rxf)                //if rxf is low and
                                                        begin
                                                                rd <= 1;
                                                                state <= WAIT2;        //an
                                                        end

                                        WAIT2:
                                                if(~rxf)                //double check
                                                        begin
                                                                rd <= 1;
                                                                state <= WAIT3;
                                                        end
                                                else
                                                        state <= WAIT;

                                        WAIT3:
                                                if(~rxf)                //and triple check
                                                        begin
                                                                rd <= 0;
                                                                state <= DATA_COMING;
                                                        end
                                                else
                                                        state <= WAIT;

                                        DATA_COMING:                //once rd goes low we g
                                                state <= DATA_COMING_2;

                                        DATA_COMING_2:
                                                state <= DATA_COMING_3;

                                        DATA_COMING_3:
                                                state <= DATA_HERE;

                                        DATA_HERE:
                                                begin
                                                        out <= data;        //the data is v
                                                        state <= DATA_LEAVING;
                                                        newout <= 1;        //let folks kno
                                                end

                                        DATA_LEAVING:                        //wait a cycle
                                                begin
                                                        //rd <= 1; // ORIGINAL
                                                        state <= DATA_LEAVING_2;
                                                        newout <= 0;        //let folks kno
                                                end

                                        DATA_LEAVING_2:                //wait another cycle
                                                state <= DATA_LEAVING_3;

                                        DATA_LEAVING_3:                //wait another cycle
                                                state <= DATA_LEAVING_4;

                                        DATA_LEAVING_4:                //wait another cycle
                                                state <= DATA_LEAVING_5;

                                        DATA_LEAVING_5:                //wait another cycle
                                                state <= DATA_LEAVING_6;

                                        DATA_LEAVING_6:                //wait another cycle
                                                begin
                                                        state <= WAIT;
                                                        rd <= 1;
                                                end
                                        default:
                                                state <= WAIT;
                                endcase
```

```
                        end
endmodule
```

A.1.13 usb_transfer_script.py

```
#! /usr/bin/env python

import serial
import wave
import struct
import scipy


’’’
6.111 USB transfer script
Luis Fernandez

Script runs through a coe file (basically row after row of 8 bit values) and
sends line by line.
’’’

ser = serial.Serial(port=’/dev/tty.usbserial-FTDHKA57’)

a = open(’audio_convert/Fa48k8bit.coe’,’r’)

for line in a:

    line = line.rstrip()[0:-1]
    line = int(line, base=2)

    b = struct.pack("<H", line)

    r = ser.write(b[0])

ser.close()
```

A.2 Labkit

A.2.1 labkit.v

```
‘default_nettype none
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
```

```
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
```

```
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

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

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

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

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

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

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

               clock_feedback_out, clock_feedback_in,

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

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

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

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

               switch,

               led,

               user1, user2, user3, user4,

               daughtercard,

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

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

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

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

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

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

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

```

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

    input   clock_feedback_in;
    output  clock_feedback_out;

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

    output  rs232_txd, rs232_rts;
    input  rs232_rxd, rs232_cts;

    input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

    input  clock_27mhz, clock1, clock2;

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

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

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

    inout [43:0] daughtercard;

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

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

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

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

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

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

// Video Input
//assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b1;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b1;
//assign tv_in_reset_b = 1'b0;
assign tv_in_clock = clock_27mhz;//1'b0;
//assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
```

```
  assign ram1_cen_b = 1’b1;
  assign ram1_ce_b = 1’b1;
  assign ram1_oe_b = 1’b1;
  assign ram1_we_b = 1’b1;
  assign ram1_bwe_b = 4’hF;
  assign clock_feedback_out = 1’b0;
  // clock_feedback_in is an input

  // Flash ROM
  // assign flash_data = 16’hZ;
  // assign flash_address = 24’h0;
  // assign flash_ce_b = 1’b1;
  // assign flash_oe_b = 1’b1;
  // assign flash_we_b = 1’b1;
  // assign flash_reset_b = 1’b0;
  // assign flash_byte_b = 1’b1;
  // // flash_sts is an input

  // RS-232 Interface
  assign rs232_txd = 1’b1;
  assign rs232_rts = 1’b1;
  // rs232_rxd and rs232_cts are inputs

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


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

  // User I/Os
  assign user1[21:4] = 28’hZ;
  assign user2 = 32’hZ;
  assign user3 = 32’hZ;
  assign user4 = 32’hZ;

  // Daughtercard Connectors
  assign daughtercard = 44’hZ;

  // SystemACE Microprocessor Port
  assign systemace_data = 16’hZ;
  assign systemace_address = 7’h0;
  assign systemace_ce_b = 1’b1;
  assign systemace_we_b = 1’b1;
```

```
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

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

////////////////////////////////////////////////////////////////////////////
  //
  // Reset Generation
  //
  // A shift register primitive is used to generate an active-high reset
  // signal that remains high for 16 clock cycles after configuration finishes
  // and the FPGA's internal clocks begin toggling.
  //
  ////////////////////////////////////////////////////////////////////////////
wire reset;
SRL16 reset_sr(.D(1'b0), .CLK(clock_27mhz), .Q(reset),
         .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

///////////////////////////////////////////////////////////////////////////////////////////
// create clocks
// use FPGA's digital clock manager to produce a 50 MHz clock
// this clock is our system clock
// to drive VGA at 640x480 (60 Hz), we need a 25 MHz vga clock
// credits to Jose for computing the required clock values
// and use of ramclock module
///////////////////////////////////////////////////////////////////////////////////////////
wire sys_clk_unbuf, sys_clk, vga_clk, vga_clk_unbuf;
wire slow_clk;
wire clk_locked;
DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(sys_clk_unbuf));
// synthesis attribute CLKFX_DIVIDE of vclk1 is 15
// synthesis attribute CLKFX_MULTIPLY of vclk1 is 28
// synthesis attribute CLK_FEEDBACK of vclk1 is NONE
// synthesis attribute CLKIN_PERIOD of vclk1 is 37
BUFG vclk2(.O(sys_clk),.I(sys_clk_unbuf));
DCM int_dcm(.CLKIN(sys_clk), .CLKFX(vga_clk_unbuf), .LOCKED(clk_locked));
// synthesis attribute CLKFX_DIVIDE of int_dcm is 4
// synthesis attribute CLKFX_MULTIPLY of int_dcm is 2
// synthesis attribute CLK_FEEDBACK of int_dcm is NONE
// synthesis attribute CLKIN_PERIOD of int_dcm is 20
BUFG int_dcm2(.O(vga_clk), .I(vga_clk_unbuf));
// assign led[7] = ~clk_locked;
// assign led[5:1] = {6{1'b1}};
slow_clk slow(.clk(sys_clk),
            .slow_clk(slow_clk));
// assign led[6] = ~slow_clk;
wire[6:0] percent_kept;
assign led[7:1] = ~percent_kept;
wire busy;
assign led[0] = ~busy;

///////////////////////////////////////////////////////////////////////////////////////////
// create debounced buttons
///////////////////////////////////////////////////////////////////////////////////////////
wire btn_up_clean, btn_down_clean, btn_left_clean, btn_right_clean;
wire btn_up_sw, btn_down_sw, btn_left_sw, btn_right_sw;
debounce btn_up_debounce(.reset(reset), .clock(clock_27mhz), .noisy(button_up), .clean(btn_u
debounce btn_down_debounce(.reset(reset), .clock(clock_27mhz), .noisy(button_down), .clean(b
debounce btn_left_debounce(.reset(reset), .clock(clock_27mhz), .noisy(button_left), .clean(b
debounce btn_right_debounce(.reset(reset), .clock(clock_27mhz), .noisy(button_right), .clean
assign btn_up_sw = ~btn_up_clean;
assign btn_down_sw = ~btn_down_clean;
assign btn_left_sw = ~btn_left_clean;
assign btn_right_sw = ~btn_right_clean;


///////////////////////////////////////////////////////////////////////////////////////////
// create switches
///////////////////////////////////////////////////////////////////////////////////////////
wire override_sw;
wire[1:0] quad_corner_sw;
assign override_sw = switch[7];
assign quad_corner_sw = switch[1:0];

///////////////////////////////////////////////////////////////////////////////////////////
// instantiate vga
///////////////////////////////////////////////////////////////////////////////////////////
wire[9:0] hcount;
wire[9:0] vcount;
wire vsync, hsync, blank;
vga vga(.vclock(vga_clk),
        .hcount(hcount),
        .vcount(vcount),
        .vsync(vsync),
        .hsync(hsync),
        .blank(blank));

reg old_btn_up, old_btn_down, old_btn_left, old_btn_right;
always @(posedge vsync) begin
        old_btn_up <= btn_up_sw;
        old_btn_down <= btn_down_sw;
        old_btn_left <= btn_left_sw;
        old_btn_right <= btn_right_sw;
end


///////////////////////////////////////////////////////////////////////////////////////////
// instantiate accel_lut and move_cursor
// essentially, corners of quadrilateral logic
///////////////////////////////////////////////////////////////////////////////////////////
wire acc_ready;
wire signed [15:0] acc_x;
wire signed [15:0] acc_y;
reg signed [15:0] acc_x_reg;
reg signed [15:0] acc_y_reg;
acc a(.clk(sys_clk), .sdo(user1[0]), .reset(reset),
.ncs(user1[1]), .sda(user1[2]), .scl(user1[3]),
.ready(acc_ready), .x(acc_x), .y(acc_y));

always @(posedge slow_clk) begin
        acc_x_reg <= acc_ready ? acc_x : 0;
        acc_y_reg <= acc_ready ? acc_y : 0;
end

wire[11:0] accel_val;
wire[75:0] quad_corners;
wire[9:0] x1_raw;
wire[8:0] y1_raw;
wire[9:0] x2_raw;
wire[8:0] y2_raw;
wire[9:0] x3_raw;
wire[8:0] y3_raw;
wire[9:0] x4_raw;
wire[8:0] y4_raw;
wire[9:0] x1;
wire[8:0] y1;
wire[9:0] x2;
wire[8:0] y2;
wire[9:0] x3;
```

```
wire[8:0] y3;
wire[9:0] x4;
wire[8:0] y4;
wire[9:0] display_x;
wire[8:0] display_y;
assign accel_val = {~acc_x_reg[15], acc_x_reg[7:4], 1’b0,
        ~acc_y_reg[15], acc_y_reg[7:4], 1’b0};
accel_lut accel_lut(.clk(slow_clk),
                .accel_val(accel_val),
                .quad_corners(quad_corners));
assign y4_raw = quad_corners[8:0];
assign x4_raw = quad_corners[18:9];
assign y3_raw = quad_corners[27:19];
assign x3_raw = quad_corners[37:28];
assign y2_raw = quad_corners[46:38];
assign x2_raw = quad_corners[56:47];
assign y1_raw = quad_corners[65:57];
assign x1_raw = quad_corners[75:66];
move_cursor move_cursor(.clk(vsync),
                    .up(btn_up_sw & ~old_btn_up),
                    .down(btn_down_sw & ~old_btn_down),
                    .left(btn_left_sw & ~old_btn_left),
                    .right(btn_right_sw & ~old_btn_right),
                    .override(override_sw),
                    .switch(quad_corner_sw),
                    .x1_raw(x1_raw),
                    .y1_raw(y1_raw),
                    .x2_raw(x2_raw),
                    .y2_raw(y2_raw),
                    .x3_raw(x3_raw),
                    .y3_raw(y3_raw),
                    .x4_raw(x4_raw),
                    .y4_raw(y4_raw),
                    .x1(x1),
                    .y1(y1),
                    .x2(x2),
                    .y2(y2),
                    .x3(x3),
                    .y3(y3),
                    .x4(x4),
                    .y4(y4),
                    .display_x(display_x),
                    .display_y(display_y));


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

```
// instantiate pixels_kept module
////////////////////////////////////////////////////////////////////////////////////////////
pixels_kept pixels_kept(
                .x1(x1),
                .y1(y1),
                .x2(x2),
                .y2(y2),
                .x3(x3),
                .y3(y3),
                .x4(x4),
                .y4(y4),
                .percent_kept(percent_kept));


////////////////////////////////////////////////////////////////////////////////////////////
// instantiate perspective_params module
////////////////////////////////////////////////////////////////////////////////////////////
wire signed[67:0] p1_inv;
wire signed[68:0] p2_inv;
wire signed[78:0] p3_inv;
wire signed[67:0] p4_inv;
wire signed[68:0] p5_inv;
wire signed[78:0] p6_inv;
wire signed[58:0] p7_inv;
wire signed[59:0] p8_inv;
wire signed[70:0] p9_inv;
wire signed[78:0] dec_numx_horiz;
wire signed[78:0] dec_numy_horiz;
wire signed[70:0] dec_denom_horiz;

perspective_params perspective_params(.clk(slow_clk),
                                    .x1(x1),
                                    .y1(y1),
                                    .x2(x2),
                                    .y2(y2),
                                    .x3(x3),
                                    .y3(y3),
                                    .x4(x4),
                                    .y4(y4),
                                    .p1_inv(p1_inv),
                                    .p2_inv(p2_inv),
                                    .p3_inv(p3_inv),
                                    .p4_inv(p4_inv),
                                    .p5_inv(p5_inv),
                                    .p6_inv(p6_inv),
                                    .p7_inv(p7_inv),
                                    .p8_inv(p8_inv),
                                    .p9_inv(p9_inv),
                                    .dec_numx_horiz(dec_numx_horiz),
                                    .dec_numy_horiz(dec_numy_horiz),
                                    .dec_denom_horiz(dec_denom_horiz));



///////////////////////////////////////////////////////////////////////////////////////////
// instantiate bram blocks
///////////////////////////////////////////////////////////////////////////////////////////

// declarations of necessary stuff
reg[16:0] ntsc_cb_in_addr = 0;
wire[16:0] ntsc_out_addr;
wire[16:0] vga_in_addr;
wire[16:0] vga_out_addr;
wire[11:0] ntsc_cb_din;
wire[11:0] ntsc_dout;
wire[11:0] vga_din;
wire[11:0] vga_dout;
wire ntsc_cb_in_wr;
wire vga_in_wr;
assign ntsc_cb_in_wr = 1;

// ntsc

adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz),
                         .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                         .tv_in_i2c_clock(tv_in_i2c_clock),
                         .tv_in_i2c_data(tv_in_i2c_data));

wire [29:0] ycrcb;        // video data (luminance, chrominance)
wire [2:0] fvh;        // sync for field, vertical, horizontal
wire       dv;        // data valid

ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                         .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                         .ycrcb(ycrcb), .f(fvh[2]),
                         .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

// code to write NTSC data to video memory

wire [16:0] ntsc_addr;
wire [11:0] ntsc_data;
wire        ntsc_we;
```

```
ntsc_to_bram n2b (tv_in_line_clock1, tv_in_line_clock1, fvh, dv,
                 ycrcb, ntsc_addr, ntsc_data, ntsc_we, switch[6]);

// dump a checkerboard into "ntsc" buffer
reg[9:0] cur_x = 0;
reg[9:0] cur_y = 0;
wire[2:0] checkerboard;
assign checkerboard = cur_x[7:5] + cur_y[7:5];
assign ntsc_cb_din = {{4{checkerboard[2]}}, {4{checkerboard[1]}}, {4{checkerboard[0]}}};
always @(posedge tv_in_line_clock1) begin
    ntsc_cb_in_addr <= (ntsc_cb_in_addr < 76799) ? (ntsc_cb_in_addr + 1) : 0;
    cur_x <= (cur_x < 319) ? (cur_x + 1) : 0;
    if ((cur_x == 319) && (cur_y == 239)) begin
        cur_y <= 0;
    end
    else if (cur_x == 319) begin
        cur_y <= cur_y + 1;
    end
end

// instantiate the pixel_map module
pixel_map pixel_map(.clk(sys_clk),
                .p1_inv(p1_inv),
                .p2_inv(p2_inv),
                .p3_inv(p3_inv),
                .p4_inv(p4_inv),
                .p5_inv(p5_inv),
                .p6_inv(p6_inv),
                .p7_inv(p7_inv),
                .p8_inv(p8_inv),
                .p9_inv(p9_inv),
                .dec_numx_horiz(dec_numx_horiz),
                .dec_numy_horiz(dec_numy_horiz),
                .dec_denom_horiz(dec_denom_horiz),
                .pixel_in(ntsc_dout),
                .pixel_out(vga_din),
                .ntsc_out_addr(ntsc_out_addr),
                .vga_in_wr(vga_in_wr),
                .vga_in_addr(vga_in_addr));

// read from vga buffer for display
addr_map addr_map(.hcount(hcount),
                .vcount(vcount),
                .addr(vga_out_addr));

/*always @(posedge sys_clk) begin
```

```
    vga_in_addr <= (vga_in_addr < 76799) ? (vga_in_addr + 1) : 0;
    ntsc_out_addr <= (ntsc_out_addr < 76799) ? (ntsc_out_addr + 1) : 0;
    vga_in_wr <= 1;
end*/

// create the brams
bram ntsc_buf(.a_clk(tv_in_line_clock1),
            .a_wr(switch[5] ? ntsc_cb_in_wr : ntsc_we),
            .a_addr(switch[5] ? ntsc_cb_in_addr : ntsc_addr),
            .a_din(switch[5] ? ntsc_cb_din : ntsc_data),
            .b_clk(sys_clk),
            .b_addr(ntsc_out_addr),
            .b_dout(ntsc_dout));

bram vga_buf(.a_clk(sys_clk),
            .a_wr(vga_in_wr),
            .a_addr(vga_in_addr),
            .a_din(vga_din),
            .b_clk(vga_clk),
            .b_addr(vga_out_addr),
            .b_dout(vga_dout));

///////////////////////////////////////////////////////////////////////////////////////////
// Create VGA output signals
// In order to meet the setup and hold times of AD7125, we send it ~vga_clk
///////////////////////////////////////////////////////////////////////////////////////////
assign vga_out_red = {vga_dout[11:8], 4'b0};
assign vga_out_green = {vga_dout[7:4], 4'b0};
assign vga_out_blue = {vga_dout[3:0], 4'b0};
assign vga_out_sync_b = 1'b1;    // not used
assign vga_out_blank_b = ~blank;
assign vga_out_pixel_clock = ~vga_clk;
assign vga_out_hsync = hsync;
assign vga_out_vsync = vsync;


///////////////////////////////////////////////////////////////////////////////////////////
// instantiate hex display
///////////////////////////////////////////////////////////////////////////////////////////
wire[63:0] hex_disp_data;
// lower 32 bits, keep nice separator of 0 between x, y
assign hex_disp_data[8:0] = display_y;
assign hex_disp_data[15:9] = 7'd0;
assign hex_disp_data[25:16] = display_x;
assign hex_disp_data[31:26] = 6'd0;
// higher bits, put the percent_kept
```

```verilog
assign hex_disp_data[63:32] = {10'b0, accel_val[11:6], 10'b0, accel_val[5:0]};
display_16hex display_16hex(.reset(reset),
            .clock_27mhz(clock_27mhz),
            .data(hex_disp_data),
            .disp_blank(disp_blank),
            .disp_clock(disp_clock),
            .disp_data_out(disp_data_out),
            .disp_rs(disp_rs),
            .disp_ce_b(disp_ce_b),
            .disp_reset_b(disp_reset_b));

  // AC97

  wire [7:0] from_ac97_data, to_ac97_data;
  wire ready;

  // allow user to adjust volume
  wire vup,vdown;
  reg old_vup,old_vdown;
  debounce bup(.reset(reset),.clock(clock_27mhz),.noisy(~button3),.clean(vup));
  debounce bdown(.reset(reset),.clock(clock_27mhz),.noisy(~button_down),.clean(vdown));
  reg [4:0] volume;
  always @ (posedge clock_27mhz) begin
    if (reset) volume <= 5'd8;
    else begin
      if (vup & ~old_vup & volume != 5'd31) volume <= volume+1;
      if (vdown & ~old_vdown & volume != 5'd0) volume <= volume-1;
    end
    old_vup <= vup;
    old_vdown <= vdown;
  end

  // AC97 driver
  lab5audio labaudio(clock_27mhz, reset, volume, from_ac97_data, to_ac97_data, ready,
        audio_reset_b, ac97_sdata_out, ac97_sdata_in,
        ac97_synch, ac97_bit_clock);

  // writeSwitch UP to write, DOWN to read
  wire writeSwitch;
  debounce sw7(.reset(reset),.clock(clock_27mhz),.noisy(switch[3]),.clean(writeSwitch));

  wire startSwitch;
  debounce sw6(.reset(reset),.clock(clock_27mhz),.noisy(switch[2]),.clean(startSwitch));

  wire memReset;
  debounce benter(.reset(reset),.clock(clock_27mhz),.noisy(~button_enter),.clean(memReset));

  wire audioTrigger;
  debounce b3(.reset(reset),.clock(clock_27mhz),.noisy(~button0),.clean(audioTrigger));

  wire [63:0] hexdisp;

  // Receive and Playback module
  audioManager management(
    .clock(clock_27mhz),
    .reset(memReset),

    // User I/O
    .startSwitch(startSwitch),
    .audioSelector(percent_kept),
    .writeSwitch(writeSwitch),
    // .hexdisp(hexdisp),
    .audioTrigger(audioTrigger),

    // AC97 I/O
    .ready(ready),
    .from_ac97_data(from_ac97_data),
    .to_ac97_data(to_ac97_data),

    // Flash I/O
    .flash_data(flash_data),
    .flash_address(flash_address),
    .flash_ce_b(flash_ce_b),
    .flash_oe_b(flash_oe_b),
    .flash_we_b(flash_we_b),
    .flash_reset_b(flash_reset_b),
    .flash_byte_b(flash_byte_b),
    .flash_sts(flash_sts),
    .busy(busy),

    // USB I/O
    .data(user1[31:24]), //the data pins from the USB fifo
    .rxf(user1[23]), //the rxf pin from the USB fifo
    .rd(user1[22]) //the rd pin TO the USB FIFO (OUTPUT)
  );

endmodule
```

A.2.2 labkit.ucf

```
################################################################################
#
```

```
# 6.111 FPGA Labkit -- Constraints File
#
# For Labkit Revision 004
#
#
# Created: Oct 30, 2004 from revision 003 contraints file
# Author: Nathan Ickes and Isaac Cambron
#
###############################################################################
#
# CHANGES FOR BOARD REVISION 004
#
# 1) Added signals for logic analyzer pods 2-4.
# 2) Expanded tv_in_ycrcy bus to 20 bits.
# 3) Renamed tv_out_sclk to tv_out_i2c_clock for consistency
# 4) Renamed tv_out_data to tv_out_i2c_data for consistency
# 5) Reversed disp_data_in and disp_data_out signals, so that "out" is an
#    output of the FPGA, and "in" is an input.
#
# CHANGES FOR BOARD REVISION 003
#
# 1) Combined flash chip enables into a single signal, flash_ce_b.
# 2) Moved SRAM feedback clock loop to FPGA pins AL28 (out) and AJ16 (in).
# 3) Moved rs232_rts to FPGA pin R3.
# 4) Moved flash_address<1> to AE14.
#
# CHANGES FOR BOARD REVISION 002
#
# 1) Moved ZBT_BANK1_CLK signal to pin Y9.
# 2) Moved user1<30> to J14.
# 3) Moved user3<29> to J13.
# 4) Added SRAM clock feedback loop between D15 and H15.
# 5) Renamed ram#_parity and ram#_we#_b signals.
# 6) Removed the constraint on "systemace_clock", since this net no longer
#    exists. The SystemACE clock is now hardwired to the 27MHz oscillator
#    on the PCB.
#
###############################################################################
#
# Complete change history (including bug fixes)
#
# 2007-Aug-16: Fixed revision history. (flash_address<1> was actually changed
#              to AE14 for revision 003.)
#
# 2005-Sep-09: Added missing IOSTANDARD attribute to "disp_data_out".
#
```

```
# 2005-Jan-23: Added a pullup to FLASH_STS
#
# 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
#              actually populated on the boards. (The boards support up to
#              256Mb devices, with 25 address lines.)
#
# 2005-Jan-23: Change history started.
#
###############################################################################

#
# Audio CODEC
#

NET "beep"           LOC="ac19" | IOSTANDARD=LVDCI_33;
NET "audio_reset_b"  LOC="ae18" | IOSTANDARD=LVTTL;
NET "ac97_sdata_out" LOC="ac18" | IOSTANDARD=LVDCI_33;
NET "ac97_sdata_in"  LOC="aj24";
NET "ac97_synch"     LOC="ac17" | IOSTANDARD=LVDCI_33;
NET "ac97_bit_clock" LOC="ah24";

NET "sys_clk" TNM_NET = sys_clk;
TIMESPEC TS_sys_clk = PERIOD "sys_clk" 20 ns HIGH 50%;

NET "vga_clk" TNM_NET = vga_clk;
TIMESPEC TS_vga_clk = PERIOD "vga_clk" 40 ns HIGH 50%;
#
# VGA Output
#

NET "vga_out_red<7>" LOC="ae9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<6>" LOC="ae8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<5>" LOC="ad12" | IOSTANDARD=LVTTL;
NET "vga_out_red<4>" LOC="af8"  | IOSTANDARD=LVTTL;
NET "vga_out_red<3>" LOC="af9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<2>" LOC="ag9"  | IOSTANDARD=LVTTL;
NET "vga_out_red<1>" LOC="ag10" | IOSTANDARD=LVTTL;
NET "vga_out_red<0>" LOC="af11" | IOSTANDARD=LVTTL;

NET "vga_out_green<7>" LOC="ah8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<6>" LOC="ah7"  | IOSTANDARD=LVTTL;
NET "vga_out_green<5>" LOC="aj6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<4>" LOC="ah6"  | IOSTANDARD=LVTTL;
NET "vga_out_green<3>" LOC="ad15" | IOSTANDARD=LVTTL;
NET "vga_out_green<2>" LOC="ac14" | IOSTANDARD=LVTTL;
NET "vga_out_green<1>" LOC="ag8"  | IOSTANDARD=LVTTL;
NET "vga_out_green<0>" LOC="ac12" | IOSTANDARD=LVTTL;

NET "vga_out_blue<7>" LOC="ag15" | IOSTANDARD=LVTTL;
NET "vga_out_blue<6>" LOC="ag14" | IOSTANDARD=LVTTL;
NET "vga_out_blue<5>" LOC="ag13" | IOSTANDARD=LVTTL;
NET "vga_out_blue<4>" LOC="ag12" | IOSTANDARD=LVTTL;
NET "vga_out_blue<3>" LOC="aj11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<2>" LOC="ah11" | IOSTANDARD=LVTTL;
NET "vga_out_blue<1>" LOC="aj10" | IOSTANDARD=LVTTL;
NET "vga_out_blue<0>" LOC="ah9"  | IOSTANDARD=LVTTL;

NET "vga_out_sync_b"      LOC="aj9"  | IOSTANDARD=LVTTL;
NET "vga_out_blank_b"     LOC="aj8"  | IOSTANDARD=LVTTL;
NET "vga_out_pixel_clock" LOC="ac10" | IOSTANDARD=LVTTL;
NET "vga_out_hsync"       LOC="ac13" | IOSTANDARD=LVTTL;
NET "vga_out_vsync"       LOC="ac11" | IOSTANDARD=LVTTL;

#
# Video Output
#

NET "tv_out_ycrcb<9>" LOC="p27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<8>" LOC="r27" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<7>" LOC="t29" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<6>" LOC="h26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<5>" LOC="j26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<4>" LOC="l26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<3>" LOC="m26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<2>" LOC="n26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<1>" LOC="p26" | IOSTANDARD=LVDCI_33;
NET "tv_out_ycrcb<0>" LOC="r26" | IOSTANDARD=LVDCI_33;

NET "tv_out_reset_b" LOC="g27" | IOSTANDARD=LVDCI_33;
NET "tv_out_clock" LOC="l27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_clock" LOC="j27" | IOSTANDARD=LVDCI_33;
NET "tv_out_i2c_data" LOC="h27" | IOSTANDARD=LVDCI_33;
NET "tv_out_pal_ntsc" LOC="j25" | IOSTANDARD=LVDCI_33;
NET "tv_out_hsync_b" LOC="n27" | IOSTANDARD=LVDCI_33;
NET "tv_out_vsync_b" LOC="m27" | IOSTANDARD=LVDCI_33;
NET "tv_out_blank_b" LOC="h25" | IOSTANDARD=LVDCI_33;
NET "tv_out_subcar_reset" LOC="k27" | IOSTANDARD=LVDCI_33;

#
# Video Input
#

```

```
NET "tv_in_ycrcb<19>" LOC="ag17";
NET "tv_in_ycrcb<18>" LOC="ag18";
NET "tv_in_ycrcb<17>" LOC="ag19";
NET "tv_in_ycrcb<16>" LOC="ag20";
NET "tv_in_ycrcb<15>" LOC="ae20";
NET "tv_in_ycrcb<14>" LOC="af21";
NET "tv_in_ycrcb<13>" LOC="ad20";
NET "tv_in_ycrcb<12>" LOC="ag23";
NET "tv_in_ycrcb<11>" LOC="aj26";
NET "tv_in_ycrcb<10>" LOC="ah26";
NET "tv_in_ycrcb<9>" LOC="w23";
NET "tv_in_ycrcb<8>" LOC="v23";
NET "tv_in_ycrcb<7>" LOC="u23";
NET "tv_in_ycrcb<6>" LOC="t23";
NET "tv_in_ycrcb<5>" LOC="t26";
NET "tv_in_ycrcb<4>" LOC="t24";
NET "tv_in_ycrcb<3>" LOC="r25";
NET "tv_in_ycrcb<2>" LOC="l30";
NET "tv_in_ycrcb<1>" LOC="m31";
NET "tv_in_ycrcb<0>" LOC="m30";

NET "tv_in_data_valid" LOC="ah25";
NET "tv_in_line_clock1" LOC="ad16" | IOSTANDARD=LVDCI_33;
NET "tv_in_line_clock2" LOC="ad17" | IOSTANDARD=LVDCI_33;
NET "tv_in_aef" LOC="aj23";
NET "tv_in_hff" LOC="ah23";
NET "tv_in_aff" LOC="aj22";
NET "tv_in_i2c_clock" LOC="ad21" | IOSTANDARD=LVDCI_33;
NET "tv_in_i2c_data" LOC="ad19" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_read" LOC="ac22" | IOSTANDARD=LVDCI_33;
NET "tv_in_fifo_clock" LOC="ag22" | IOSTANDARD=LVDCI_33;
NET "tv_in_iso" LOC="aj27" | IOSTANDARD=LVDCI_33;
NET "tv_in_reset_b" LOC="ag25" | IOSTANDARD=LVDCI_33;
NET "tv_in_clock" LOC="ab21" | IOSTANDARD=LVDCI_33;

#
# SRAMs
#

NET "ram0_data<35>" LOC="ab25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<34>" LOC="ah29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<33>" LOC="ag28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<32>" LOC="ag29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<31>" LOC="af27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<30>" LOC="af29" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<29>" LOC="af28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<28>" LOC="ae28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<27>" LOC="ad25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<26>" LOC="aa25" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<25>" LOC="ah30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<24>" LOC="ah31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<23>" LOC="ag30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<22>" LOC="ag31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<21>" LOC="af30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<20>" LOC="af31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<19>" LOC="ae30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<18>" LOC="ae31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<17>" LOC="y27"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<16>" LOC="aa28" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<15>" LOC="y29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<14>" LOC="y28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<13>" LOC="w29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<12>" LOC="w28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<11>" LOC="v28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<10>" LOC="u29"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<9>"  LOC="u28"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<8>"  LOC="aa27" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<7>"  LOC="ad31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<6>"  LOC="ac30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<5>"  LOC="ac31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<4>"  LOC="ab30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<3>"  LOC="ab31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<2>"  LOC="aa30" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<1>"  LOC="aa31" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram0_data<0>"  LOC="y30"  | IOSTANDARD=LVDCI_33 | NODELAY;

NET "ram0_address<18>" LOC="v31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<17>" LOC="w31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<16>" LOC="ad28" | IOSTANDARD=LVDCI_33;
NET "ram0_address<15>" LOC="ad29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<14>" LOC="ac24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<13>" LOC="ad26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<12>" LOC="ad27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<11>" LOC="ac27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<10>" LOC="ab27" | IOSTANDARD=LVDCI_33;
NET "ram0_address<9>" LOC="y31" | IOSTANDARD=LVDCI_33;
NET "ram0_address<8>" LOC="w30" | IOSTANDARD=LVDCI_33;
NET "ram0_address<7>" LOC="y26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<6>" LOC="y25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<5>" LOC="ab24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<4>" LOC="ac25" | IOSTANDARD=LVDCI_33;
NET "ram0_address<3>" LOC="aa26" | IOSTANDARD=LVDCI_33;
NET "ram0_address<2>" LOC="aa24" | IOSTANDARD=LVDCI_33;
NET "ram0_address<1>" LOC="ab29" | IOSTANDARD=LVDCI_33;
NET "ram0_address<0>" LOC="ac26" | IOSTANDARD=LVDCI_33;

NET "ram0_adv_ld" LOC="v26" | IOSTANDARD=LVDCI_33;
NET "ram0_clk" LOC="u30" | IOSTANDARD=LVDCI_33;
NET "ram0_cen_b" LOC="u25" | IOSTANDARD=LVDCI_33;
NET "ram0_ce_b" LOC="w26" | IOSTANDARD=LVDCI_33;
NET "ram0_oe_b" LOC="v25" | IOSTANDARD=LVDCI_33;
NET "ram0_we_b" LOC="u31" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<0>" LOC="v27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<1>" LOC="u27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<2>" LOC="w27" | IOSTANDARD=LVDCI_33;
NET "ram0_bwe_b<3>" LOC="u26" | IOSTANDARD=LVDCI_33;

NET "ram1_data<35>" LOC="aa9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<34>" LOC="ah2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<33>" LOC="ah1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<32>" LOC="ag2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<31>" LOC="ag1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<30>" LOC="af2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<29>" LOC="af1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<28>" LOC="ae2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<27>" LOC="ae1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<26>" LOC="ab9" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<25>" LOC="ah3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<24>" LOC="ag4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<23>" LOC="ag3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<22>" LOC="af4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<21>" LOC="af3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<20>" LOC="ae4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<19>" LOC="ae5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<18>" LOC="ad5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<17>" LOC="v2"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<16>" LOC="ad1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<15>" LOC="ac2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<14>" LOC="ac1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<13>" LOC="ab2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<12>" LOC="ab1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<11>" LOC="aa2" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<10>" LOC="aa1" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<9>"  LOC="y2"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<8>"  LOC="v4"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<7>"  LOC="ac3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<6>"  LOC="ac4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<5>"  LOC="aa5" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<4>"  LOC="aa3" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<3>"  LOC="aa4" | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<2>"  LOC="y3"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<1>"  LOC="y4"  | IOSTANDARD=LVDCI_33 | NODELAY;
NET "ram1_data<0>"  LOC="w3"  | IOSTANDARD=LVDCI_33 | NODELAY;

NET "ram1_address<18>" LOC="ab3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<17>" LOC="ac5" | IOSTANDARD=LVDCI_33;
NET "ram1_address<16>" LOC="u6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<15>" LOC="v6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<14>" LOC="w6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<13>" LOC="y6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<12>" LOC="aa7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<11>" LOC="ab7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<10>" LOC="ac6" | IOSTANDARD=LVDCI_33;
NET "ram1_address<9>" LOC="ad3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<8>" LOC="ad4" | IOSTANDARD=LVDCI_33;
NET "ram1_address<7>" LOC="u3" | IOSTANDARD=LVDCI_33;
NET "ram1_address<6>" LOC="w4" | IOSTANDARD=LVDCI_33;
NET "ram1_address<5>" LOC="ac8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<4>" LOC="ab8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<3>" LOC="aa8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<2>" LOC="y7" | IOSTANDARD=LVDCI_33;
NET "ram1_address<1>" LOC="y8" | IOSTANDARD=LVDCI_33;
NET "ram1_address<0>" LOC="ad7" | IOSTANDARD=LVDCI_33;

NET "ram1_adv_ld" LOC="y5" | IOSTANDARD=LVDCI_33;
NET "ram1_clk" LOC="y9" | IOSTANDARD=LVDCI_33;
NET "ram1_cen_b" LOC="v5" | IOSTANDARD=LVDCI_33;
NET "ram1_ce_b" LOC="u4" | IOSTANDARD=LVDCI_33;
NET "ram1_oe_b" LOC="w5" | IOSTANDARD=LVDCI_33;
NET "ram1_we_b" LOC="aa6" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<0>" LOC="u2" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<1>" LOC="u1" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<2>" LOC="v1" | IOSTANDARD=LVDCI_33;
NET "ram1_bwe_b<3>" LOC="u5" | IOSTANDARD=LVDCI_33;

NET "clock_feedback_out" LOC="al28" | IOSTANDARD=LVDCI_33;
NET "clock_feedback_in" LOC="aj16";

#
# Flash
#

NET "flash_data<15>" LOC="ak10" | IOSTANDARD=LVTTL;
NET "flash_data<14>" LOC="ak11" | IOSTANDARD=LVTTL;
NET "flash_data<13>" LOC="ak12" | IOSTANDARD=LVTTL;
NET "flash_data<12>" LOC="ak13" | IOSTANDARD=LVTTL;
NET "flash_data<11>" LOC="ak14" | IOSTANDARD=LVTTL;
NET "flash_data<10>" LOC="ak15" | IOSTANDARD=LVTTL;
NET "flash_data<9>"  LOC="ah12" | IOSTANDARD=LVTTL;
NET "flash_data<8>"  LOC="ah13" | IOSTANDARD=LVTTL;
NET "flash_data<7>"  LOC="al10" | IOSTANDARD=LVTTL;
NET "flash_data<6>"  LOC="al11" | IOSTANDARD=LVTTL;
NET "flash_data<5>"  LOC="al12" | IOSTANDARD=LVTTL;
NET "flash_data<4>"  LOC="al13" | IOSTANDARD=LVTTL;
NET "flash_data<3>"  LOC="al14" | IOSTANDARD=LVTTL;
NET "flash_data<2>"  LOC="al15" | IOSTANDARD=LVTTL;
NET "flash_data<1>"  LOC="aj12" | IOSTANDARD=LVTTL;
NET "flash_data<0>"  LOC="aj13" | IOSTANDARD=LVTTL;

NET "flash_address<24>" LOC="al7";
NET "flash_address<23>" LOC="aj15";
NET "flash_address<22>" LOC="al25";
NET "flash_address<21>" LOC="ak23";
NET "flash_address<20>" LOC="al23";
NET "flash_address<19>" LOC="ak22";
NET "flash_address<18>" LOC="al22";
NET "flash_address<17>" LOC="ak21";
NET "flash_address<16>" LOC="al21";
NET "flash_address<15>" LOC="ak20";
NET "flash_address<14>" LOC="al20";
NET "flash_address<13>" LOC="ak19";
NET "flash_address<12>" LOC="al19";
NET "flash_address<11>" LOC="al18";
NET "flash_address<10>" LOC="ak17";
NET "flash_address<9>"  LOC="al17";
NET "flash_address<8>"  LOC="ah21";
NET "flash_address<7>"  LOC="aj20";
NET "flash_address<6>"  LOC="ah20";
NET "flash_address<5>"  LOC="aj19";
NET "flash_address<4>"  LOC="ah19";
NET "flash_address<3>"  LOC="ah18";
NET "flash_address<2>"  LOC="aj17";
NET "flash_address<1>"  LOC="ae14";
NET "flash_address<0>"  LOC="ah14";

NET "flash_ce_b" LOC="aj21" | IOSTANDARD=LVDCI_33;

NET "flash_oe_b" LOC="ak9" | IOSTANDARD=LVDCI_33;
NET "flash_we_b" LOC="al8" | IOSTANDARD=LVDCI_33;
NET "flash_reset_b" LOC="ak18" | IOSTANDARD=LVDCI_33;
NET "flash_sts" LOC="al9" | PULLUP;
NET "flash_byte_b" LOC="ah15" | IOSTANDARD=LVDCI_33;

#
# RS-232
#

NET "rs232_txd" LOC="p4" | IOSTANDARD=LVDCI_33;
NET "rs232_rxd" LOC="p6";
NET "rs232_rts" LOC="r3" | IOSTANDARD=LVDCI_33;
NET "rs232_cts" LOC="n8";

#
# Mouse and Keyboard
#

NET "mouse_clock" LOC="ac16";
NET "mouse_data" LOC="ac15";
NET "keyboard_clock" LOC="ag16";
NET "keyboard_data" LOC="af16";

#
# Clocks
#

NET "clock_27mhz" LOC="c16";
NET "clock1" LOC="h16";
NET "clock2" LOC="c15";

#
# Alphanumeric Display
#

NET "disp_blank" LOC="af12" | IOSTANDARD=LVDCI_33;
NET "disp_data_in" LOC="ae12";
NET "disp_clock" LOC="af14" | IOSTANDARD=LVDCI_33;
NET "disp_rs" LOC="af15" | IOSTANDARD=LVDCI_33;
NET "disp_ce_b" LOC="af13" | IOSTANDARD=LVDCI_33;
NET "disp_reset_b" LOC="ag11" | IOSTANDARD=LVDCI_33;
NET "disp_data_out" LOC="ae15" | IOSTANDARD=LVDCI_33;

#
# Buttons and Switches
#

NET "button0" LOC="ae11";
NET "button1" LOC="ae10";
NET "button2" LOC="ad11";
NET "button3" LOC="ab12";
NET "button_enter" LOC="ak7";
NET "button_right" LOC="al6";
NET "button_left" LOC="al5";
NET "button_up" LOC="al4";
NET "button_down" LOC="ak6";

NET "switch<7>" LOC="ad22";
NET "switch<6>" LOC="ae23";
NET "switch<5>" LOC="ac20";
NET "switch<4>" LOC="ab20";
NET "switch<3>" LOC="ac21";
NET "switch<2>" LOC="ak25";
NET "switch<1>" LOC="al26";
NET "switch<0>" LOC="ak26";

#
# Discrete LEDs
#

NET "led<7>" LOC="ae17" | IOSTANDARD=LVTTL;
NET "led<6>" LOC="af17" | IOSTANDARD=LVTTL;
NET "led<5>" LOC="af18" | IOSTANDARD=LVTTL;
NET "led<4>" LOC="af19" | IOSTANDARD=LVTTL;
NET "led<3>" LOC="af20" | IOSTANDARD=LVTTL;
NET "led<2>" LOC="ag21" | IOSTANDARD=LVTTL;
NET "led<1>" LOC="ae21" | IOSTANDARD=LVTTL;
NET "led<0>" LOC="ae22" | IOSTANDARD=LVTTL;


#
# User Pins
#

NET "user1<31>" LOC="j15" | IOSTANDARD=LVTTL;
NET "user1<30>" LOC="j14" | IOSTANDARD=LVTTL;
NET "user1<29>" LOC="g15" | IOSTANDARD=LVTTL;
NET "user1<28>" LOC="f14" | IOSTANDARD=LVTTL;
NET "user1<27>" LOC="f12" | IOSTANDARD=LVTTL;
NET "user1<26>" LOC="h11" | IOSTANDARD=LVTTL;
NET "user1<25>" LOC="g9" | IOSTANDARD=LVTTL;
NET "user1<24>" LOC="h9" | IOSTANDARD=LVTTL;
NET "user1<23>" LOC="b15" | IOSTANDARD=LVTTL;
NET "user1<22>" LOC="b14" | IOSTANDARD=LVTTL;
NET "user1<21>" LOC="f15" | IOSTANDARD=LVTTL;
NET "user1<20>" LOC="e13" | IOSTANDARD=LVTTL;
NET "user1<19>" LOC="e11" | IOSTANDARD=LVTTL;
NET "user1<18>" LOC="e9" | IOSTANDARD=LVTTL;
NET "user1<17>" LOC="f8" | IOSTANDARD=LVTTL;
NET "user1<16>" LOC="f7" | IOSTANDARD=LVTTL;
NET "user1<15>" LOC="c13" | IOSTANDARD=LVTTL;
NET "user1<14>" LOC="c12" | IOSTANDARD=LVTTL;
NET "user1<13>" LOC="c11" | IOSTANDARD=LVTTL;
NET "user1<12>" LOC="c10" | IOSTANDARD=LVTTL;
NET "user1<11>" LOC="c9" | IOSTANDARD=LVTTL;
NET "user1<10>" LOC="c8" | IOSTANDARD=LVTTL;
NET "user1<9>" LOC="c6" | IOSTANDARD=LVTTL;
NET "user1<8>" LOC="e6" | IOSTANDARD=LVTTL;
NET "user1<7>" LOC="a11" | IOSTANDARD=LVTTL;
NET "user1<6>" LOC="a10" | IOSTANDARD=LVTTL;
NET "user1<5>" LOC="a9" | IOSTANDARD=LVTTL;
NET "user1<4>" LOC="a8" | IOSTANDARD=LVTTL;
NET "user1<3>" LOC="b6" | IOSTANDARD=LVTTL;
NET "user1<2>" LOC="b5" | IOSTANDARD=LVTTL;
NET "user1<1>" LOC="c5" | IOSTANDARD=LVTTL;
NET "user1<0>" LOC="b3" | IOSTANDARD=LVTTL;


NET "user2<31>" LOC="b27" | IOSTANDARD=LVTTL;
NET "user2<30>" LOC="b26" | IOSTANDARD=LVTTL;
NET "user2<29>" LOC="b25" | IOSTANDARD=LVTTL;
NET "user2<28>" LOC="a24" | IOSTANDARD=LVTTL;
NET "user2<27>" LOC="a23" | IOSTANDARD=LVTTL;
NET "user2<26>" LOC="a22" | IOSTANDARD=LVTTL;
NET "user2<25>" LOC="a21" | IOSTANDARD=LVTTL;
NET "user2<24>" LOC="a20" | IOSTANDARD=LVTTL;
NET "user2<23>" LOC="d26" | IOSTANDARD=LVTTL;
NET "user2<22>" LOC="d25" | IOSTANDARD=LVTTL;
NET "user2<21>" LOC="c24" | IOSTANDARD=LVTTL;
NET "user2<20>" LOC="d23" | IOSTANDARD=LVTTL;
NET "user2<19>" LOC="d21" | IOSTANDARD=LVTTL;
NET "user2<18>" LOC="d20" | IOSTANDARD=LVTTL;
NET "user2<17>" LOC="d19" | IOSTANDARD=LVTTL;
NET "user2<16>" LOC="d18" | IOSTANDARD=LVTTL;
NET "user2<15>" LOC="f24" | IOSTANDARD=LVTTL;
NET "user2<14>" LOC="f23" | IOSTANDARD=LVTTL;
NET "user2<13>" LOC="e22" | IOSTANDARD=LVTTL;
NET "user2<12>" LOC="e20" | IOSTANDARD=LVTTL;
NET "user2<11>" LOC="e18" | IOSTANDARD=LVTTL;
NET "user2<10>" LOC="e16" | IOSTANDARD=LVTTL;
NET "user2<9>" LOC="a19" | IOSTANDARD=LVTTL;
NET "user2<8>" LOC="a18" | IOSTANDARD=LVTTL;
NET "user2<7>" LOC="h22" | IOSTANDARD=LVTTL;
NET "user2<6>" LOC="g22" | IOSTANDARD=LVTTL;
NET "user2<5>" LOC="f21" | IOSTANDARD=LVTTL;
NET "user2<4>" LOC="f19" | IOSTANDARD=LVTTL;
NET "user2<3>" LOC="f17" | IOSTANDARD=LVTTL;
NET "user2<2>" LOC="h19" | IOSTANDARD=LVTTL;
NET "user2<1>" LOC="g20" | IOSTANDARD=LVTTL;
NET "user2<0>" LOC="h21" | IOSTANDARD=LVTTL;

NET "user3<31>" LOC="g12" | IOSTANDARD=LVTTL;
NET "user3<30>" LOC="h13" | IOSTANDARD=LVTTL;
NET "user3<29>" LOC="j13" | IOSTANDARD=LVTTL;
NET "user3<28>" LOC="g14" | IOSTANDARD=LVTTL;
NET "user3<27>" LOC="f13" | IOSTANDARD=LVTTL;
NET "user3<26>" LOC="f11" | IOSTANDARD=LVTTL;
NET "user3<25>" LOC="g10" | IOSTANDARD=LVTTL;
NET "user3<24>" LOC="h10" | IOSTANDARD=LVTTL;
NET "user3<23>" LOC="a15" | IOSTANDARD=LVTTL;
NET "user3<22>" LOC="a14" | IOSTANDARD=LVTTL;
NET "user3<21>" LOC="e15" | IOSTANDARD=LVTTL;
NET "user3<20>" LOC="e14" | IOSTANDARD=LVTTL;
NET "user3<19>" LOC="e12" | IOSTANDARD=LVTTL;
NET "user3<18>" LOC="e10" | IOSTANDARD=LVTTL;
NET "user3<17>" LOC="f9" | IOSTANDARD=LVTTL;
NET "user3<16>" LOC="g8" | IOSTANDARD=LVTTL;
NET "user3<15>" LOC="d14" | IOSTANDARD=LVTTL;
NET "user3<14>" LOC="d13" | IOSTANDARD=LVTTL;
NET "user3<13>" LOC="d12" | IOSTANDARD=LVTTL;
NET "user3<12>" LOC="d11" | IOSTANDARD=LVTTL;
NET "user3<11>" LOC="d9" | IOSTANDARD=LVTTL;
NET "user3<10>" LOC="d8" | IOSTANDARD=LVTTL;
NET "user3<9>" LOC="d7" | IOSTANDARD=LVTTL;
NET "user3<8>" LOC="d6" | IOSTANDARD=LVTTL;
NET "user3<7>" LOC="b12" | IOSTANDARD=LVTTL;
NET "user3<6>" LOC="b11" | IOSTANDARD=LVTTL;
NET "user3<5>" LOC="b10" | IOSTANDARD=LVTTL;
NET "user3<4>" LOC="b9" | IOSTANDARD=LVTTL;
NET "user3<3>" LOC="a7" | IOSTANDARD=LVTTL;
NET "user3<2>" LOC="a6" | IOSTANDARD=LVTTL;
NET "user3<1>" LOC="a5" | IOSTANDARD=LVTTL;
NET "user3<0>" LOC="a4" | IOSTANDARD=LVTTL;

NET "user4<31>" LOC="a28" | IOSTANDARD=LVTTL;
NET "user4<30>" LOC="a27" | IOSTANDARD=LVTTL;
NET "user4<29>" LOC="a26" | IOSTANDARD=LVTTL;
NET "user4<28>" LOC="a25" | IOSTANDARD=LVTTL;
NET "user4<27>" LOC="b23" | IOSTANDARD=LVTTL;
NET "user4<26>" LOC="b22" | IOSTANDARD=LVTTL;
NET "user4<25>" LOC="b21" | IOSTANDARD=LVTTL;
NET "user4<24>" LOC="b20" | IOSTANDARD=LVTTL;
NET "user4<23>" LOC="e25" | IOSTANDARD=LVTTL;
NET "user4<22>" LOC="c26" | IOSTANDARD=LVTTL;
NET "user4<21>" LOC="d24" | IOSTANDARD=LVTTL;
NET "user4<20>" LOC="c23" | IOSTANDARD=LVTTL;
NET "user4<19>" LOC="c22" | IOSTANDARD=LVTTL;
NET "user4<18>" LOC="c21" | IOSTANDARD=LVTTL;
NET "user4<17>" LOC="c20" | IOSTANDARD=LVTTL;
NET "user4<16>" LOC="c19" | IOSTANDARD=LVTTL;
NET "user4<15>" LOC="g24" | IOSTANDARD=LVTTL;
NET "user4<14>" LOC="e24" | IOSTANDARD=LVTTL;
NET "user4<13>" LOC="e23" | IOSTANDARD=LVTTL;
NET "user4<12>" LOC="e21" | IOSTANDARD=LVTTL;
NET "user4<11>" LOC="e19" | IOSTANDARD=LVTTL;
NET "user4<10>" LOC="e17" | IOSTANDARD=LVTTL;
NET "user4<9>" LOC="b19" | IOSTANDARD=LVTTL;
NET "user4<8>" LOC="b18" | IOSTANDARD=LVTTL;
NET "user4<7>" LOC="h23" | IOSTANDARD=LVTTL;
NET "user4<6>" LOC="g23" | IOSTANDARD=LVTTL;
NET "user4<5>" LOC="g21" | IOSTANDARD=LVTTL;
NET "user4<4>" LOC="f20" | IOSTANDARD=LVTTL;
NET "user4<3>" LOC="f18" | IOSTANDARD=LVTTL;
NET "user4<2>" LOC="f16" | IOSTANDARD=LVTTL;
NET "user4<1>" LOC="g18" | IOSTANDARD=LVTTL;
NET "user4<0>" LOC="g17" | IOSTANDARD=LVTTL;

#
# Daughter Card
#

NET "daughtercard<43>" LOC="L7" | IOSTANDARD=LVTTL;
NET "daughtercard<42>" LOC="H1" | IOSTANDARD=LVTTL;
NET "daughtercard<41>" LOC="J2" | IOSTANDARD=LVTTL;
NET "daughtercard<40>" LOC="J1" | IOSTANDARD=LVTTL;
NET "daughtercard<39>" LOC="K2" | IOSTANDARD=LVTTL;
NET "daughtercard<38>" LOC="M7" | IOSTANDARD=LVTTL;
NET "daughtercard<37>" LOC="M6" | IOSTANDARD=LVTTL;
NET "daughtercard<36>" LOC="M3" | IOSTANDARD=LVTTL;
NET "daughtercard<35>" LOC="M4" | IOSTANDARD=LVTTL;
NET "daughtercard<34>" LOC="L3" | IOSTANDARD=LVTTL;
NET "daughtercard<33>" LOC="K1" | IOSTANDARD=LVTTL;
NET "daughtercard<32>" LOC="L4" | IOSTANDARD=LVTTL;
NET "daughtercard<31>" LOC="K3" | IOSTANDARD=LVTTL;
NET "daughtercard<30>" LOC="K9" | IOSTANDARD=LVTTL;
NET "daughtercard<29>" LOC="L9" | IOSTANDARD=LVTTL;
NET "daughtercard<28>" LOC="K8" | IOSTANDARD=LVTTL;
NET "daughtercard<27>" LOC="K7" | IOSTANDARD=LVTTL;
NET "daughtercard<26>" LOC="L8" | IOSTANDARD=LVTTL;
NET "daughtercard<25>" LOC="L6" | IOSTANDARD=LVTTL;
NET "daughtercard<24>" LOC="M5" | IOSTANDARD=LVTTL;
NET "daughtercard<23>" LOC="N5" | IOSTANDARD=LVTTL;
NET "daughtercard<22>" LOC="P5" | IOSTANDARD=LVTTL;
NET "daughtercard<21>" LOC="D3" | IOSTANDARD=LVTTL;
NET "daughtercard<20>" LOC="E4" | IOSTANDARD=LVTTL;
NET "daughtercard<19>" LOC="E3" | IOSTANDARD=LVTTL;
NET "daughtercard<18>" LOC="F4" | IOSTANDARD=LVTTL;
NET "daughtercard<17>" LOC="F3" | IOSTANDARD=LVTTL;
NET "daughtercard<16>" LOC="G4" | IOSTANDARD=LVTTL;
NET "daughtercard<15>" LOC="H4" | IOSTANDARD=LVTTL;
NET "daughtercard<14>" LOC="J3" | IOSTANDARD=LVTTL;
NET "daughtercard<13>" LOC="J4" | IOSTANDARD=LVTTL;
NET "daughtercard<12>" LOC="D2" | IOSTANDARD=LVTTL;
NET "daughtercard<11>" LOC="D1" | IOSTANDARD=LVTTL;
NET "daughtercard<10>" LOC="E2" | IOSTANDARD=LVTTL;
NET "daughtercard<9>" LOC="E1" | IOSTANDARD=LVTTL;
NET "daughtercard<8>" LOC="F5" | IOSTANDARD=LVTTL;
NET "daughtercard<7>" LOC="G5" | IOSTANDARD=LVTTL;
NET "daughtercard<6>" LOC="H5" | IOSTANDARD=LVTTL;
NET "daughtercard<5>" LOC="J5" | IOSTANDARD=LVTTL;
NET "daughtercard<4>" LOC="K5" | IOSTANDARD=LVTTL;
NET "daughtercard<3>" LOC="H7" | IOSTANDARD=LVTTL;
NET "daughtercard<2>" LOC="J8" | IOSTANDARD=LVTTL;
NET "daughtercard<1>" LOC="J6" | IOSTANDARD=LVTTL;
NET "daughtercard<0>" LOC="J7" | IOSTANDARD=LVTTL;

#
# System Ace
#

NET "systemace_data<15>" LOC="F29" | IOSTANDARD=LVTTL;
NET "systemace_data<14>" LOC="G28" | IOSTANDARD=LVTTL;
NET "systemace_data<13>" LOC="H29" | IOSTANDARD=LVTTL;
NET "systemace_data<12>" LOC="H28" | IOSTANDARD=LVTTL;
NET "systemace_data<11>" LOC="J29" | IOSTANDARD=LVTTL;
NET "systemace_data<10>" LOC="J28" | IOSTANDARD=LVTTL;
NET "systemace_data<9>" LOC="K29" | IOSTANDARD=LVTTL;
NET "systemace_data<8>" LOC="L29" | IOSTANDARD=LVTTL;
NET "systemace_data<7>" LOC="L28" | IOSTANDARD=LVTTL;
NET "systemace_data<6>" LOC="M29" | IOSTANDARD=LVTTL;
NET "systemace_data<5>" LOC="M28" | IOSTANDARD=LVTTL;
NET "systemace_data<4>" LOC="N29" | IOSTANDARD=LVTTL;
NET "systemace_data<3>" LOC="N28" | IOSTANDARD=LVTTL;
NET "systemace_data<2>" LOC="P28" | IOSTANDARD=LVTTL;
NET "systemace_data<1>" LOC="R29" | IOSTANDARD=LVTTL;
NET "systemace_data<0>" LOC="R28" | IOSTANDARD=LVTTL;

NET "systemace_address<6>" LOC="E29" | IOSTANDARD=LVTTL;
NET "systemace_address<5>" LOC="F28" | IOSTANDARD=LVTTL;
NET "systemace_address<4>" LOC="H31" | IOSTANDARD=LVTTL;
NET "systemace_address<3>" LOC="J30" | IOSTANDARD=LVTTL;
NET "systemace_address<2>" LOC="J31" | IOSTANDARD=LVTTL;
NET "systemace_address<1>" LOC="K30" | IOSTANDARD=LVTTL;
NET "systemace_address<0>" LOC="K31" | IOSTANDARD=LVTTL;

NET "systemace_ce_b" LOC="E28" | IOSTANDARD=LVTTL;
NET "systemace_we_b" LOC="P31" | IOSTANDARD=LVTTL;
NET "systemace_oe_b" LOC="R31" | IOSTANDARD=LVTTL;
NET "systemace_irq" LOC="D29";
NET "systemace_mpbrdy" LOC="L31";

#
# Logic Analyzer
#

NET "analyzer1_data<15>" LOC="G1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<14>" LOC="H3" | IOSTANDARD=LVTTL;
NET "analyzer1_data<13>" LOC="M9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<12>" LOC="M8" | IOSTANDARD=LVTTL;
NET "analyzer1_data<11>" LOC="L5" | IOSTANDARD=LVTTL;
NET "analyzer1_data<10>" LOC="L1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<9>" LOC="L2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<8>" LOC="N9" | IOSTANDARD=LVTTL;
NET "analyzer1_data<7>" LOC="M1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<6>" LOC="M2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<5>" LOC="N1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<4>" LOC="N2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<3>" LOC="P1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<2>" LOC="P2" | IOSTANDARD=LVTTL;
NET "analyzer1_data<1>" LOC="R1" | IOSTANDARD=LVTTL;
NET "analyzer1_data<0>" LOC="R2" | IOSTANDARD=LVTTL;
NET "analyzer1_clock" LOC="G2" | IOSTANDARD=LVTTL;

NET "analyzer2_data<15>" LOC="f2" | IOSTANDARD=LVTTL;
NET "analyzer2_data<14>" LOC="k10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<13>" LOC="l10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<12>" LOC="m10" | IOSTANDARD=LVTTL;
NET "analyzer2_data<11>" LOC="r7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<10>" LOC="n3" | IOSTANDARD=LVTTL;
NET "analyzer2_data<9>" LOC="r8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<8>" LOC="r9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<7>" LOC="p9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<6>" LOC="n6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<5>" LOC="p7" | IOSTANDARD=LVTTL;
NET "analyzer2_data<4>" LOC="n4" | IOSTANDARD=LVTTL;
NET "analyzer2_data<3>" LOC="t8" | IOSTANDARD=LVTTL;
NET "analyzer2_data<2>" LOC="t9" | IOSTANDARD=LVTTL;
NET "analyzer2_data<1>" LOC="r6" | IOSTANDARD=LVTTL;
NET "analyzer2_data<0>" LOC="r5" | IOSTANDARD=LVTTL;
NET "analyzer2_clock" LOC="f1" | IOSTANDARD=LVTTL;

NET "analyzer3_data<15>" LOC="k24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<14>" LOC="k25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<13>" LOC="k22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<12>" LOC="l24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<11>" LOC="l25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<10>" LOC="l22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<9>" LOC="l23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<8>" LOC="m23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<7>" LOC="m24" | IOSTANDARD=LVTTL;
NET "analyzer3_data<6>" LOC="m25" | IOSTANDARD=LVTTL;
NET "analyzer3_data<5>" LOC="k23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<4>" LOC="m22" | IOSTANDARD=LVTTL;
NET "analyzer3_data<3>" LOC="n23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<2>" LOC="p23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<1>" LOC="r23" | IOSTANDARD=LVTTL;
NET "analyzer3_data<0>" LOC="r24" | IOSTANDARD=LVTTL;
NET "analyzer3_clock" LOC="j24" | IOSTANDARD=LVTTL;

NET "analyzer4_data<15>" LOC="ag7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<14>" LOC="ak3" | IOSTANDARD=LVTTL;
NET "analyzer4_data<13>" LOC="aj5" | IOSTANDARD=LVTTL;
NET "analyzer4_data<12>" LOC="ak29" | IOSTANDARD=LVTTL;
NET "analyzer4_data<11>" LOC="ak28" | IOSTANDARD=LVTTL;
NET "analyzer4_data<10>" LOC="af25" | IOSTANDARD=LVTTL;
NET "analyzer4_data<9>" LOC="ag24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<8>" LOC="af24" | IOSTANDARD=LVTTL;
NET "analyzer4_data<7>" LOC="af23" | IOSTANDARD=LVTTL;
NET "analyzer4_data<6>" LOC="al27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<5>" LOC="ak27" | IOSTANDARD=LVTTL;
NET "analyzer4_data<4>" LOC="ah17" | IOSTANDARD=LVTTL;
NET "analyzer4_data<3>" LOC="ad13" | IOSTANDARD=LVTTL;
NET "analyzer4_data<2>" LOC="v7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<1>" LOC="u7" | IOSTANDARD=LVTTL;
NET "analyzer4_data<0>" LOC="u8" | IOSTANDARD=LVTTL;
NET "analyzer4_clock" LOC="ad9" | IOSTANDARD=LVTTL;
```

A.3 Our Modules

A.3.1 acc.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
/*
Copyright (C) {2014}  {James Thomas} <jamesjoethomas@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
////////////////////////////////////////////////////////////////////////////////
/*
output starts on same cycle start is asserted
high-order bits produced first
input only needs to be supplied on the cycle that start is high
produces zeros if input exhausted
assume W > 1
*/
module par_to_ser #(parameter W=8)
        (input clk /* device clock */, input[W-1:0] par,
        input start, output ser);
        reg[W-1:0] par_reg = 0;

        always @(posedge clk) begin
                if (start) begin
                        par_reg <= {par[W-2:0], 1'b0};
                end
                else begin
                        par_reg <= {par_reg[W-2:0], 1’b0};
                end
        end

        assign ser = start ? par[W-1] : par_reg[W-1];
endmodule

/*
output appears W-1 clock cycles after first serial bit sent
assume high-order bits are input first
assume W > 2
*/
module ser_to_par #(parameter W=8)
        (input clk /* device clock */, input ser,
        output[W-1:0] par);
        reg[W-2:0] par_reg = 0;

        always @(posedge clk) begin
                par_reg <= {par_reg[W-3:0], ser};
        end

        assign par = {par_reg, ser};
endmodule

/*
reduces the system clock by a factor of 6
*/
module acc_clk(input clk /* system clock */, output dev_clk);
        parameter TICKS = 9;

        reg [3:0] count = 0;
        reg sig_reg = 0;

        always @(posedge clk) begin
                if (count == TICKS) begin
                        // flip at half period
                        sig_reg <= ~sig_reg;
                        count <= 0;
                end
                else begin
                        count <= count + 1;
                end
        end
        assign dev_clk = sig_reg;
endmodule
```

```verilog

/*
assert in_ready when a new datapoint is available, avg_ready will
be signalled after 32 data points have been folded into the average
*/
module moving_avg(
  input clock, in_ready, reset,
  input signed [15:0] data,
  output signed [15:0] avg,
  output avg_ready
);
        // circular buffer
        reg signed [15:0] samples [31:0];

        reg [4:0] offset = 0;
        reg signed [15:0] accum = 0;
        reg [5:0] num_samples = 0;
        reg signed [15:0] data_right_shift;

        always @(*) begin
                data_right_shift = {data[15], data[15], data[15], data[15],
                        data[15], data[15:5]};
        end

        always @(posedge clock) begin
                if (reset) begin
                        accum <= 0;
                        num_samples <= 0;
                        offset <= 0;
                end
                else if (in_ready) begin
                        num_samples <= (num_samples == 6'd32) ? num_samples : num_samples +
                        samples[offset] <= data_right_shift;
                        if (num_samples == 6'd32) begin
                                accum <= accum + data_right_shift - samples[offset];
                        end
                        else begin
                                accum <= accum + data_right_shift;
                        end
                        offset <= offset + 1;
                end
        end

  assign avg = accum;
  assign avg_ready = (num_samples == 6'd32) ? 1 : 0;
endmodule
```

```

/*
ready permanently asserted after initialization completed
acc operates completely with the slowed accelerometer clock
*/
module acc(input clk /* system clock */, sdo, reset,
        output ncs, sda, scl, ready, output signed [15:0] x, y);
        // TODO use state machine -- transition through all of the initiatialization states
        // register), then rotate through the value reading states
        // one cycle gap between states to allow for CS deassertion
        parameter MEASURE_INIT = 0;
        parameter X_READ = 1;
        parameter Y_READ = 2;
        reg[1:0] state = MEASURE_INIT; // TODO: set the right number of bits for this
        reg[4:0] count = 0; // TODO: set the right number of bits for this

        reg ncs_reg;

        wire dev_clk;
        acc_clk ac(.clk(clk), .dev_clk(dev_clk));

        reg[7:0] par_in;
        reg pts_start;
        par_to_ser pts(.clk(dev_clk), .par(par_in), .start(pts_start), .ser(sda));

        wire[7:0] par_out;
        ser_to_par stp(.clk(dev_clk), .ser(sdo), .par(par_out));

        reg ma_x_in_ready;
        reg [7:0] x_low_bits = 0;
        reg signed [15:0] ma_x_in;
        wire ma_x_avg_ready;
        wire signed [15:0] ma_x_avg;
        moving_avg ma_x(
          .clock(dev_clk), .in_ready(ma_x_in_ready), .reset(reset),
          .data(ma_x_in),
          .avg(ma_x_avg),
          .avg_ready(ma_x_avg_ready)
        );

        reg ma_y_in_ready;
        reg [7:0] y_low_bits = 0;
        reg signed [15:0] ma_y_in;
        wire ma_y_avg_ready;
        wire signed [15:0] ma_y_avg;
        moving_avg ma_y(
          .clock(dev_clk), .in_ready(ma_y_in_ready), .reset(reset),
          .data(ma_y_in),
          .avg(ma_y_avg),
          .avg_ready(ma_y_avg_ready)
        );

        // invariants: when transitioning out of a state always set counter to 0
        always @(posedge dev_clk) begin
                case (state)
                        MEASURE_INIT: begin
                                if (count == 5’d18) begin
                                        count <= 0;
                                        state <= X_READ;
                                end
                                else begin
                                        count <= count + 1;
                                end
                        end
                        X_READ: begin
                                if (count == 5’d25) begin
                                        count <= 0;
                                        state <= Y_READ;
                                end
                                else begin
                                        count <= count + 1;
                                end
                                if (count == 5’d17) begin
                                        x_low_bits <= par_out;
                                end
                        end
                        Y_READ: begin
                                if (count == 5’d25) begin
                                        count <= 0;
                                        state <= X_READ;
                                end
                                else begin
                                        count <= count + 1;
                                end
                                if (count == 5’d17) begin
                                        y_low_bits <= par_out;
                                end
                        end
                endcase
        end

        always @(*) begin
```

```verilog
                case (state)
                        MEASURE_INIT: begin
                                pts_start = (count == 5'd2 || count == 5'd10) ? 1 : 0;
                                if (count == 5'd2) begin
                                        par_in = 8'h2D; // 0 for W, 0 for MB
                                end
                                else if (count == 5'd10) begin
                                        par_in = 8'h08; // set measure bit
                                end
                                else begin
                                        par_in = 0;
                                end
                                ma_x_in_ready = 0; ma_x_in = 0;
                                ma_y_in_ready = 0; ma_y_in = 0;
                                ncs_reg = (count == 5'd18 || count == 5'd0) ? 1 : 0;
                        end
                        X_READ: begin
                                pts_start = (count == 5'd1) ? 1 : 0;
                                par_in = (count == 5'd1) ? 8'hF2 : 0; // 1 for R, 1 for MB
                                ma_x_in_ready = (count == 5'd25) ? 1 : 0;
                                ma_x_in = (count == 5'd25) ? {par_out, x_low_bits} : 0;
                                ma_y_in_ready = 0; ma_y_in = 0;
                                ncs_reg = (count == 5'd25) ? 1 : 0;
                        end
                        Y_READ: begin
                                pts_start = (count == 5'd1) ? 1 : 0;
                                par_in = (count == 5'd1) ? 8'hF4 : 0; // 1 for R, 1 for MB
                                ma_y_in_ready = (count == 5'd25) ? 1 : 0;
                                ma_y_in = (count == 5'd25) ? {par_out, y_low_bits} : 0;
                                ma_x_in_ready = 0; ma_x_in = 0;
                                ncs_reg = (count == 5'd25) ? 1 : 0;
                        end
                endcase
        end


        assign scl = (ncs_reg == 1 || (state == MEASURE_INIT && count == 5'd1
                || state != MEASURE_INIT && count == 5'd0)) ? 1 : dev_clk;
        assign ncs = ncs_reg;
        assign ready = ma_x_avg_ready && ma_y_avg_ready;
        assign x = ma_x_avg;
        assign y = ma_y_avg;
endmodule
```

A.3.2 accel_lut.v

```
//////////////////////////////////////////////////////////////////////////////////
//This file was autogenerated by accel_lut.jl.
//DO NOT MANUALLY EDIT THIS FILE!!!

//This file implements accel_lut rom for lookup of quadrilateral corners
//based on accelerometer readings
//////////////////////////////////////////////////////////////////////////////////

module accel_lut(input clk, input[11:0] accel_val, output reg[75:0] quad_corners);
always @(posedge clk) begin
        case (accel_val)
                12'd0: quad_corners = 76'd16690381550355664320;
                12'd1: quad_corners = 76'd16690381550355664320;
                12'd2: quad_corners = 76'd16690381550355664320;
                12'd3: quad_corners = 76'd16690381550355664320;
                12'd4: quad_corners = 76'd16690381550355664320;
                12'd5: quad_corners = 76'd16690381550355664320;
                12'd6: quad_corners = 76'd16690381550355664320;
                12'd7: quad_corners = 76'd16690381550355664320;
                12'd8: quad_corners = 76'd16690381550355664320;
                12'd9: quad_corners = 76'd16690381550355664320;
                12'd10: quad_corners = 76'd16690381550355664320;
                12'd11: quad_corners = 76'd16690381550355664320;
                12'd12: quad_corners = 76'd16690381550355664320;
                12'd13: quad_corners = 76'd16690381550355664320;
                12'd14: quad_corners = 76'd16690381550355664320;
                12'd15: quad_corners = 76'd16690381550355664320;
                12'd16: quad_corners = 76'd16690381550355664320;
                12'd17: quad_corners = 76'd16690381550355664320;
                12'd18: quad_corners = 76'd16690381550355664320;
                12'd19: quad_corners = 76'd16690381550355664320;
                12'd20: quad_corners = 76'd16690381550355664320;
                12'd21: quad_corners = 76'd16690381550355664320;
                12'd22: quad_corners = 76'd16690381550355664320;
                12'd23: quad_corners = 76'd16690381550355664320;
                12'd24: quad_corners = 76'd16690381550355664320;
                12'd25: quad_corners = 76'd16675913736552686l312;
                12'd26: quad_corners = 76'd92683508482071876096;
                12'd27: quad_corners = 76'd9253883034404259737 6;
                12'd28: quad_corners = 76'd9239429294350115020 8;
                12'd29: quad_corners = 76'd9224961480547187148 9;
                12'd30: quad_corners = 76'd9210493666744259277 0;
                12'd31: quad_corners = 76'd9196039899202376243 5;
                12'd32: quad_corners = 76'd16545858113558624256 5;
```

```
            12'd33: quad_corners = 76'd165313902172117936646;
```

```
                12’d79: quad_corners = 76’d16690381550355664320;
                12’d80: quad_corners = 76’d16690381550355664320;
                12’d81: quad_corners = 76’d16690381550355664320;
                12’d82: quad_corners = 76’d16690381550355664320;
                12’d83: quad_corners = 76’d16690381550355664320;
                12’d84: quad_corners = 76’d16690381550355664320;
                12’d85: quad_corners = 76’d16690381550355664320;
                12’d86: quad_corners = 76’d16690381550355664320;
                12’d87: quad_corners = 76’d16690381550355664320;
                12’d88: quad_corners = 76’d16690381550355664320;
                12’d89: quad_corners = 76’d16675913736552686131 2;
                12’d90: quad_corners = 76’d9268350848207187609 6;
                12'd125: quad_corners = 76'd15637958438387834906 19;
```

```
        12'd171: quad_corners = 76'd680087433370416177180;
        12'd172: quad_corners = 76'd753729730151224956958;
        12'd173: quad_corners = 76'd827227771281079520801;
        12'd174: quad_corners = 76'd900870068061619865125;
        12'd175: quad_corners = 76'd1048299200674119531560;
        12'd176: quad_corners = 76'd1121941356717172044332;
        12'd177: quad_corners = 76'd1195439397571879741487;
        12'd178: quad_corners = 76'd1342868670646990380595;
        12'd179: quad_corners = 76'd1490297802984343704631;
        12'd180: quad_corners = 76'd1563795843838783490619;
        12'd181: quad_corners = 76'd1563795843838783490619;
        12'd182: quad_corners = 76'd1563795843838783490619;
        12'd183: quad_corners = 76'd1563795843838783490619;
        12'd184: quad_corners = 76'd1563795843838783490619;
        12'd185: quad_corners = 76'd1563795843838783490619;
        12'd186: quad_corners = 76'd1563795843838783490619;
        12'd187: quad_corners = 76'd1563795843838783490619;
        12'd188: quad_corners = 76'd1563795843838783490619;
        12'd189: quad_corners = 76'd1563795843838783490619;
        12'd190: quad_corners = 76'd1563795843838783490619;
        12'd191: quad_corners = 76'd1563795843838783490619;
        12'd192: quad_corners = 76'd166903815503556664320;
        12'd193: quad_corners = 76'd166903815503556664320;
        12'd194: quad_corners = 76'd166903815503556664320;
        12'd195: quad_corners = 76'd166903815503556664320;
        12'd196: quad_corners = 76'd166903815503556664320;
        12'd197: quad_corners = 76'd166903815503556664320;
        12'd198: quad_corners = 76'd166903815503556664320;
        12'd199: quad_corners = 76'd166903815503556664320;
        12'd200: quad_corners = 76'd166903815503556664320;
        12'd201: quad_corners = 76'd166903815503556664320;
        12'd202: quad_corners = 76'd166903815503556664320;
        12'd203: quad_corners = 76'd166903815503556664320;
        12'd204: quad_corners = 76'd166903815503556664320;
        12'd205: quad_corners = 76'd166903815503556664320;
        12'd206: quad_corners = 76'd166903815503556664320;
        12'd207: quad_corners = 76'd166903815503556664320;
        12'd208: quad_corners = 76'd166903815503556664320;
        12'd209: quad_corners = 76'd166903815503556664320;
        12'd210: quad_corners = 76'd166903815503556664320;
        12'd211: quad_corners = 76'd166903815503556664320;
        12'd212: quad_corners = 76'd166903815503556664320;
        12'd213: quad_corners = 76'd166903815503556664320;
        12'd214: quad_corners = 76'd166903815503556664320;
        12'd215: quad_corners = 76'd166903815503556664320;
        12'd216: quad_corners = 76'd166903815503556664320;
```

```
            12'd217: quad_corners = 76'd166759137365526861312;
            12'd263: quad_corners = 76'd16690381550355666432O;
            12'd309: quad_corners = 76'd15637958438387834906l9;
            12'd355: quad_corners = 76'd238811520539482660361;
            12'd356: quad_corners = 76'd238666841575745918475;
            12'd357: quad_corners = 76'd312165023443625184781;
            12'd358: quad_corners = 76'd312020344479620007951;
            12'd359: quad_corners = 76'd385662500798086774801;
            12'd360: quad_corners = 76'd459304797854041896980;
            12'd361: quad_corners = 76'd532802979996530634774;
            12'd362: quad_corners = 76'd606445136314729490457;
            12'd363: quad_corners = 76'd680087433370416177180;
            12'd364: quad_corners = 76'd753729730151224956958;
            12'd365: quad_corners = 76'd827227771281079520801;
            12'd366: quad_corners = 76'd900870068061619865125;
            12'd367: quad_corners = 76'd1048299200674119531560;
            12'd368: quad_corners = 76'd1121941356717172044332;
            12'd369: quad_corners = 76'd1195439397571879741487;
            12'd370: quad_corners = 76'd1342868670646990380595;
            12'd371: quad_corners = 76'd1490297802984343704631;
            12'd372: quad_corners = 76'd1563795843838783490619;
            12'd373: quad_corners = 76'd1563795843838783490619;
            12'd374: quad_corners = 76'd1563795843838783490619;
            12'd375: quad_corners = 76'd1563795843838783490619;
            12'd376: quad_corners = 76'd1563795843838783490619;
            12'd377: quad_corners = 76'd1563795843838783490619;
            12'd378: quad_corners = 76'd1563795843838783490619;
            12'd379: quad_corners = 76'd1563795843838783490619;
            12'd380: quad_corners = 76'd1563795843838783490619;
            12'd381: quad_corners = 76'd1563795843838783490619;
            12'd382: quad_corners = 76'd1563795843838783490619;
            12'd383: quad_corners = 76'd1563795843838783490619;
            12'd384: quad_corners = 76'd166903815503556664320;
            12'd385: quad_corners = 76'd166903815503556664320;
            12'd386: quad_corners = 76'd166903815503556664320;
            12'd387: quad_corners = 76'd166903815503556664320;
            12'd388: quad_corners = 76'd166903815503556664320;
            12'd389: quad_corners = 76'd166903815503556664320;
            12'd390: quad_corners = 76'd166903815503556664320;
            12'd391: quad_corners = 76'd166903815503556664320;
            12'd392: quad_corners = 76'd166903815503556664320;
            12'd393: quad_corners = 76'd166903815503556664320;
            12'd394: quad_corners = 76'd166903815503556664320;
            12'd395: quad_corners = 76'd166903815503556664320;
            12'd396: quad_corners = 76'd166903815503556664320;
            12'd397: quad_corners = 76'd166903815503556664320;
            12'd398: quad_corners = 76'd166903815503556664320;
            12'd399: quad_corners = 76'd166903815503556664320;
            12'd400: quad_corners = 76'd166903815503556664320;
            12’d401: quad_corners = 76’d166903815503556664320;
            12’d402: quad_corners = 76’d166903815503556664320;
            12’d403: quad_corners = 76’d166903815503556664320;
            12’d404: quad_corners = 76’d166903815503556664320;
            12’d405: quad_corners = 76’d166903815503556664320;
            12’d406: quad_corners = 76’d166903815503556664320;
            12’d407: quad_corners = 76’d166903815503556664320;
            12’d408: quad_corners = 76’d166903815503556664320;
            12’d409: quad_corners = 76’d166759137365526861312;
            12’d410: quad_corners = 76’d92683508482071876096;
            12’d411: quad_corners = 76’d92538830344042597376;
            12’d412: quad_corners = 76’d92394292943501150208;
            12’d413: quad_corners = 76’d92249614805471871489;
            12’d414: quad_corners = 76’d92104936667442592770;
            12’d415: quad_corners = 76’d91960398992023762435;
            12’d416: quad_corners = 76’d165458581135586242565;
            12’d417: quad_corners = 76’d165313902172117936646;
            12’d418: quad_corners = 76’d165169223208381195272;
            12’d419: quad_corners = 76’d238811520539482660361;
            12’d420: quad_corners = 76’d238666841575745918475;
            12’d421: quad_corners = 76’d312165023443625184781;
            12’d422: quad_corners = 76’d312020344479620007951;
            12’d423: quad_corners = 76’d385662500798086774801;
            12’d424: quad_corners = 76’d459304797854041896980;
            12’d425: quad_corners = 76’d532802979996530634774;
            12’d426: quad_corners = 76’d606445136314729490457;
            12’d427: quad_corners = 76’d680087433370416177180;
            12’d428: quad_corners = 76’d753729730151224956958;
            12’d429: quad_corners = 76’d827227771281079520801;
            12’d430: quad_corners = 76’d900870068061619865125;
            12’d431: quad_corners = 76’d1048299200674119531560;
            12’d432: quad_corners = 76’d1121941356717172044332;
            12’d433: quad_corners = 76’d1195439397571879741487;
            12’d434: quad_corners = 76’d1342868670646990380595;
            12’d435: quad_corners = 76’d1490297802984343704631;
            12’d436: quad_corners = 76’d1563795843838783490619;
            12’d437: quad_corners = 76’d1563795843838783490619;
            12’d438: quad_corners = 76’d1563795843838783490619;
            12’d439: quad_corners = 76’d1563795843838783490619;
            12’d440: quad_corners = 76’d1563795843838783490619;
            12’d441: quad_corners = 76’d1563795843838783490619;
            12’d442: quad_corners = 76’d1563795843838783490619;
            12’d443: quad_corners = 76’d1563795843838783490619;
            12’d444: quad_corners = 76’d1563795843838783490619;
            12’d445: quad_corners = 76’d1563795843838783490619;
            12’d446: quad_corners = 76’d1563795843838783490619;
            12'd447: quad_corners = 76'd1563795843838783490619;
```

```
                12’d493: quad_corners = 76’d827227771281079520801;
                12’d494: quad_corners = 76’d900870068061619865125;
                12’d495: quad_corners = 76’d1048299200674119531560;
                12’d496: quad_corners = 76’d1121941356717172044332;
                12’d497: quad_corners = 76’d1195439397571879741487;
                12’d498: quad_corners = 76’d1342868670646990380595;
                12’d499: quad_corners = 76’d1490297802984343704631;
                12’d500: quad_corners = 76’d1563795843838783490619;
                12’d501: quad_corners = 76’d1563795843838783490619;
                12’d502: quad_corners = 76’d1563795843838783490619;
                12’d503: quad_corners = 76’d1563795843838783490619;
                12’d504: quad_corners = 76’d1563795843838783490619;
                12’d505: quad_corners = 76’d1563795843838783490619;
                12’d506: quad_corners = 76’d1563795843838783490619;
                12’d507: quad_corners = 76’d1563795843838783490619;
                12’d508: quad_corners = 76’d1563795843838783490619;
                12’d509: quad_corners = 76’d1563795843838783490619;
                12’d510: quad_corners = 76’d1563795843838783490619;
                12’d511: quad_corners = 76’d1563795843838783490619;
                12’d512: quad_corners = 76’d166903815503556664320;
                12’d513: quad_corners = 76’d166903815503556664320;
                12’d514: quad_corners = 76’d166903815503556664320;
                12’d515: quad_corners = 76’d166903815503556664320;
                12’d516: quad_corners = 76’d166903815503556664320;
                12’d517: quad_corners = 76’d166903815503556664320;
                12’d518: quad_corners = 76’d166903815503556664320;
                12’d519: quad_corners = 76’d166903815503556664320;
                12’d520: quad_corners = 76’d166903815503556664320;
                12’d521: quad_corners = 76’d166903815503556664320;
                12’d522: quad_corners = 76’d166903815503556664320;
                12’d523: quad_corners = 76’d166903815503556664320;
                12’d524: quad_corners = 76’d166903815503556664320;
                12’d525: quad_corners = 76’d166903815503556664320;
                12’d526: quad_corners = 76’d166903815503556664320;
                12’d527: quad_corners = 76’d166903815503556664320;
                12’d528: quad_corners = 76’d166903815503556664320;
                12’d529: quad_corners = 76’d166903815503556664320;
                12’d530: quad_corners = 76’d166903815503556664320;
                12’d531: quad_corners = 76’d166903815503556664320;
                12’d532: quad_corners = 76’d166903815503556664320;
                12’d533: quad_corners = 76’d166903815503556664320;
                12’d534: quad_corners = 76’d166903815503556664320;
                12’d535: quad_corners = 76’d166903815503556664320;
                12’d536: quad_corners = 76’d166903815503556664320;
                12’d537: quad_corners = 76’d166759137365526861312;
                12’d538: quad_corners = 76’d92683508482071876096;
                12'd539: quad_corners = 76'd92538830344042597376;
                12'd540: quad_corners = 76'd92394292943501150208;
                12'd541: quad_corners = 76'd92249614805471871489;
                12'd542: quad_corners = 76'd92104936667442592770;
                12'd543: quad_corners = 76'd91960398992023762435;
                12'd544: quad_corners = 76'd165458581135586242565;
                12'd545: quad_corners = 76'd165313902172117936646;
                12'd546: quad_corners = 76'd165169223208381195272;
                12'd547: quad_corners = 76'd238811520539482660361;
                12'd548: quad_corners = 76'd238666841575745918475;
                12'd549: quad_corners = 76'd312165023443625184781;
                12'd550: quad_corners = 76'd312020344479620007951;
                12'd551: quad_corners = 76'd385662500798086774801;
                12'd552: quad_corners = 76'd459304797854041896980;
                12'd553: quad_corners = 76'd532802979996530634774;
                12'd554: quad_corners = 76'd606445136314729490457;
                12'd555: quad_corners = 76'd680087433370416177180;
                12'd556: quad_corners = 76'd753729730151224956958;
                12'd557: quad_corners = 76'd827227771281079520801;
                12'd558: quad_corners = 76'd900870068061619865125;
                12'd559: quad_corners = 76'd1048299200674119531560;
                12'd560: quad_corners = 76'd1121941356717172044332;
                12'd561: quad_corners = 76'd1195439397571879741487;
                12'd562: quad_corners = 76'd1342868670646990380595;
                12'd563: quad_corners = 76'd1490297802984343704631;
                12'd564: quad_corners = 76'd1563795843838783490619;
                12'd565: quad_corners = 76'd1563795843838783490619;
                12'd566: quad_corners = 76'd1563795843838783490619;
                12'd567: quad_corners = 76'd1563795843838783490619;
                12'd568: quad_corners = 76'd1563795843838783490619;
                12'd569: quad_corners = 76'd1563795843838783490619;
                12'd570: quad_corners = 76'd1563795843838783490619;
                12'd571: quad_corners = 76'd1563795843838783490619;
                12'd572: quad_corners = 76'd1563795843838783490619;
                12'd573: quad_corners = 76'd1563795843838783490619;
                12'd574: quad_corners = 76'd1563795843838783490619;
                12'd575: quad_corners = 76'd1563795843838783490619;
                12'd576: quad_corners = 76'd166903815503556664320;
                12'd577: quad_corners = 76'd166903815503556664320;
                12'd578: quad_corners = 76'd166903815503556664320;
                12'd579: quad_corners = 76'd166903815503556664320;
                12'd580: quad_corners = 76'd166903815503556664320;
                12'd581: quad_corners = 76'd166903815503556664320;
                12'd582: quad_corners = 76'd166903815503556664320;
                12'd583: quad_corners = 76'd166903815503556664320;
                12'd584: quad_corners = 76'd166903815503556664320;
```

```
            12'd585: quad_corners = 76'd166903815503556664320;
            12'd586: quad_corners = 76'd166903815503556664320;
            12'd587: quad_corners = 76'd166903815503556664320;
            12'd588: quad_corners = 76'd166903815503556664320;
            12'd589: quad_corners = 76'd166903815503556664320;
            12'd590: quad_corners = 76'd166903815503556664320;
            12'd591: quad_corners = 76'd166903815503556664320;
            12'd592: quad_corners = 76'd166903815503556664320;
            12'd593: quad_corners = 76'd166903815503556664320;
            12'd594: quad_corners = 76'd166903815503556664320;
            12'd595: quad_corners = 76'd166903815503556664320;
            12'd596: quad_corners = 76'd166903815503556664320;
            12'd597: quad_corners = 76'd166903815503556664320;
            12'd598: quad_corners = 76'd166903815503556664320;
            12'd599: quad_corners = 76'd166903815503556664320;
            12'd600: quad_corners = 76'd166903815503556664320;
            12'd601: quad_corners = 76'd166759137365526861312;
            12'd602: quad_corners = 76'd92683508482071876096;
            12'd603: quad_corners = 76'd92538830344042597376;
            12'd604: quad_corners = 76'd92394292943501150208;
            12'd605: quad_corners = 76'd92249614805471871489;
            12'd606: quad_corners = 76'd92104936667442592770;
            12'd607: quad_corners = 76'd91960398992023762435;
            12'd608: quad_corners = 76'd165458581135586242565;
            12'd609: quad_corners = 76'd165313902172117936646;
            12'd610: quad_corners = 76'd165169223208381195272;
            12'd611: quad_corners = 76'd238811520539482660361;
            12'd612: quad_corners = 76'd238666841575745918475;
            12'd613: quad_corners = 76'd312165023443625184781;
            12'd614: quad_corners = 76'd312020344479620007951;
            12'd615: quad_corners = 76'd385662500798086774801;
            12'd616: quad_corners = 76'd459304797854041896980;
            12'd617: quad_corners = 76'd532802979996530634774;
            12'd618: quad_corners = 76'd606445136314729490457;
            12'd619: quad_corners = 76'd680087433370416177180;
            12'd620: quad_corners = 76'd753729730151224956958;
            12'd621: quad_corners = 76'd827227771281079520801;
            12'd622: quad_corners = 76'd900870068061619865125;
            12'd623: quad_corners = 76'd1048299200674119531560;
            12'd624: quad_corners = 76'd1121941356717172044332;
            12'd625: quad_corners = 76'd1195439397571879741487;
            12'd626: quad_corners = 76'd1342868670646990380595;
            12'd627: quad_corners = 76'd1490297802984343704631;
            12'd628: quad_corners = 76'd1563795843838783490619;
            12'd629: quad_corners = 76'd1563795843838783490619;
            12'd630: quad_corners = 76'd1563795843838783490619;
            12’d631: quad_corners = 76’d15637958438387834906l9;
            12'd677: quad_corners = 76'd312165023443625184781;
            12'd678: quad_corners = 76'd312020344479620007951;
            12'd679: quad_corners = 76'd385662500798086774801;
            12'd680: quad_corners = 76'd459304797854041896980;
            12'd681: quad_corners = 76'd532802979996530634774;
            12'd682: quad_corners = 76'd606445136314729490457;
            12'd683: quad_corners = 76'd680087433370416177180;
            12'd684: quad_corners = 76'd753729730151224956958;
            12'd685: quad_corners = 76'd827227771281079520801;
            12'd686: quad_corners = 76'd900870068061619865125;
            12'd687: quad_corners = 76'd1048299200674119531560;
            12'd688: quad_corners = 76'd1121941356717172044332;
            12'd689: quad_corners = 76'd1195439397571879741487;
            12'd690: quad_corners = 76'd1342868670646990380595;
            12'd691: quad_corners = 76'd1490297802984343704631;
            12'd692: quad_corners = 76'd1563795843838783490619;
            12'd693: quad_corners = 76'd1563795843838783490619;
            12'd694: quad_corners = 76'd1563795843838783490619;
            12'd695: quad_corners = 76'd1563795843838783490619;
            12'd696: quad_corners = 76'd1563795843838783490619;
            12'd697: quad_corners = 76'd1563795843838783490619;
            12'd698: quad_corners = 76'd1563795843838783490619;
            12'd699: quad_corners = 76'd1563795843838783490619;
            12'd700: quad_corners = 76'd1563795843838783490619;
            12'd701: quad_corners = 76'd1563795843838783490619;
            12'd702: quad_corners = 76'd1563795843838783490619;
            12'd703: quad_corners = 76'd1563795843838783490619;
            12'd704: quad_corners = 76'd166903815503556664320;
            12'd705: quad_corners = 76'd166903815503556664320;
            12'd706: quad_corners = 76'd166903815503556664320;
            12'd707: quad_corners = 76'd166903815503556664320;
            12'd708: quad_corners = 76'd166903815503556664320;
            12'd709: quad_corners = 76'd166903815503556664320;
            12'd710: quad_corners = 76'd166903815503556664320;
            12'd711: quad_corners = 76'd166903815503556664320;
            12'd712: quad_corners = 76'd166903815503556664320;
            12'd713: quad_corners = 76'd166903815503556664320;
            12'd714: quad_corners = 76'd166903815503556664320;
            12'd715: quad_corners = 76'd166903815503556664320;
            12'd716: quad_corners = 76'd166903815503556664320;
            12'd717: quad_corners = 76'd166903815503556664320;
            12'd718: quad_corners = 76'd166903815503556664320;
            12'd719: quad_corners = 76'd166903815503556664320;
            12'd720: quad_corners = 76'd166903815503556664320;
            12'd721: quad_corners = 76'd166903815503556664320;
            12'd722: quad_corners = 76'd166903815503556664320;
            12’d723: quad_corners = 76’d166903815503556664320;
            12’d724: quad_corners = 76’d166903815503556664320;
            12’d725: quad_corners = 76’d166903815503556664320;
            12’d726: quad_corners = 76’d166903815503556664320;
            12’d727: quad_corners = 76’d166903815503556664320;
            12’d728: quad_corners = 76’d166903815503556664320;
            12’d729: quad_corners = 76’d166759137365526861312;
            12’d730: quad_corners = 76’d92683508482071876096;
            12’d731: quad_corners = 76’d92538830344042597376;
            12’d732: quad_corners = 76’d92394292943501150208;
            12’d733: quad_corners = 76’d92249614805471871489;
            12’d734: quad_corners = 76’d92104936667442592770;
            12’d735: quad_corners = 76’d91960398992023762435;
            12’d736: quad_corners = 76’d165458581135586242565;
            12’d737: quad_corners = 76’d165313902172117936646;
            12’d738: quad_corners = 76’d165169223208381195272;
            12’d739: quad_corners = 76’d238811520539482660361;
            12’d740: quad_corners = 76’d238666841575745918475;
            12’d741: quad_corners = 76’d312165023443625184781;
            12’d742: quad_corners = 76’d312020344479620007951;
            12’d743: quad_corners = 76’d385662500798086774801;
            12’d744: quad_corners = 76’d459304797854041896980;
            12’d745: quad_corners = 76’d532802979996530634774;
            12’d746: quad_corners = 76’d606445136314729490457;
            12’d747: quad_corners = 76’d680087433370416177180;
            12’d748: quad_corners = 76’d753729730151224956958;
            12’d749: quad_corners = 76’d827227771281079520801;
            12’d750: quad_corners = 76’d900870068061619865125;
            12’d751: quad_corners = 76’d1048299200674119531560;
            12’d752: quad_corners = 76’d1121941356717172044332;
            12’d753: quad_corners = 76’d1195439397571879741487;
            12’d754: quad_corners = 76’d1342868670646990380595;
            12’d755: quad_corners = 76’d1490297802984343704631;
            12’d756: quad_corners = 76’d1563795843838783490619;
            12’d757: quad_corners = 76’d1563795843838783490619;
            12’d758: quad_corners = 76’d1563795843838783490619;
            12’d759: quad_corners = 76’d1563795843838783490619;
            12’d760: quad_corners = 76’d1563795843838783490619;
            12’d761: quad_corners = 76’d1563795843838783490619;
            12’d762: quad_corners = 76’d1563795843838783490619;
            12’d763: quad_corners = 76’d1563795843838783490619;
            12’d764: quad_corners = 76’d1563795843838783490619;
            12’d765: quad_corners = 76’d1563795843838783490619;
            12’d766: quad_corners = 76’d1563795843838783490619;
            12’d767: quad_corners = 76’d1563795843838783490619;
            12’d768: quad_corners = 76’d166903815503556664320;
```

```
                12'd769: quad_corners = 76'd16690381550355666432O;
```

```
            12'd815: quad_corners = 76'd1048299200674119531560;
            12'd816: quad_corners = 76'd1121941356717172044332;
            12'd817: quad_corners = 76'd1195439397571879741487;
            12'd818: quad_corners = 76'd1342868670646990380595;
            12'd819: quad_corners = 76'd1490297802984343704631;
            12'd820: quad_corners = 76'd1563795843838783490619;
            12'd821: quad_corners = 76'd1563795843838783490619;
            12'd822: quad_corners = 76'd1563795843838783490619;
            12'd823: quad_corners = 76'd1563795843838783490619;
            12'd824: quad_corners = 76'd1563795843838783490619;
            12'd825: quad_corners = 76'd1563795843838783490619;
            12'd826: quad_corners = 76'd1563795843838783490619;
            12'd827: quad_corners = 76'd1563795843838783490619;
            12'd828: quad_corners = 76'd1563795843838783490619;
            12'd829: quad_corners = 76'd1563795843838783490619;
            12'd830: quad_corners = 76'd1563795843838783490619;
            12'd831: quad_corners = 76'd1563795843838783490619;
            12'd832: quad_corners = 76'd166903815503556664320;
            12'd833: quad_corners = 76'd166903815503556664320;
            12'd834: quad_corners = 76'd166903815503556664320;
            12'd835: quad_corners = 76'd166903815503556664320;
            12'd836: quad_corners = 76'd166903815503556664320;
            12'd837: quad_corners = 76'd166903815503556664320;
            12'd838: quad_corners = 76'd166903815503556664320;
            12'd839: quad_corners = 76'd166903815503556664320;
            12'd840: quad_corners = 76'd166903815503556664320;
            12'd841: quad_corners = 76'd166903815503556664320;
            12'd842: quad_corners = 76'd166903815503556664320;
            12'd843: quad_corners = 76'd166903815503556664320;
            12'd844: quad_corners = 76'd166903815503556664320;
            12'd845: quad_corners = 76'd166903815503556664320;
            12'd846: quad_corners = 76'd166903815503556664320;
            12'd847: quad_corners = 76'd166903815503556664320;
            12'd848: quad_corners = 76'd166903815503556664320;
            12'd849: quad_corners = 76'd166903815503556664320;
            12'd850: quad_corners = 76'd166903815503556664320;
            12'd851: quad_corners = 76'd166903815503556664320;
            12'd852: quad_corners = 76'd166903815503556664320;
            12'd853: quad_corners = 76'd166903815503556664320;
            12'd854: quad_corners = 76'd166903815503556664320;
            12'd855: quad_corners = 76'd166903815503556664320;
            12'd856: quad_corners = 76'd166903815503556664320;
            12'd857: quad_corners = 76'd166759137365526861312;
            12'd858: quad_corners = 76'd92683508482071876096;
            12'd859: quad_corners = 76'd92538830344042597376;
            12'd860: quad_corners = 76'd92394292943501150208;
            12'd861: quad_corners = 76'd92249614805471871489;
            12'd862: quad_corners = 76'd92104936667442592770;
            12'd863: quad_corners = 76'd91960398992023762435;
            12'd864: quad_corners = 76'd165458581135586242565;
            12'd865: quad_corners = 76'd165313902172117936646;
            12'd866: quad_corners = 76'd165169223208381195272;
            12'd867: quad_corners = 76'd238811520539482660361;
            12'd868: quad_corners = 76'd238666841575745918475;
            12'd869: quad_corners = 76'd312165023443625184781;
            12'd870: quad_corners = 76'd312020344479620007951;
            12'd871: quad_corners = 76'd385662500798086774801;
            12'd872: quad_corners = 76'd459304797854041896980;
            12'd873: quad_corners = 76'd532802979996530634774;
            12'd874: quad_corners = 76'd606445136314729490457;
            12'd875: quad_corners = 76'd680087433370416177180;
            12'd876: quad_corners = 76'd753729730151224956958;
            12'd877: quad_corners = 76'd827227771281079520801;
            12'd878: quad_corners = 76'd900870068061619865125;
            12'd879: quad_corners = 76'd1048299200674119531560;
            12'd880: quad_corners = 76'd1121941356717172044332;
            12'd881: quad_corners = 76'd1195439397571879741487;
            12'd882: quad_corners = 76'd1342868670646990380595;
            12'd883: quad_corners = 76'd1490297802984343704631;
            12'd884: quad_corners = 76'd1563795843838783490619;
            12'd885: quad_corners = 76'd1563795843838783490619;
            12'd886: quad_corners = 76'd1563795843838783490619;
            12'd887: quad_corners = 76'd1563795843838783490619;
            12'd888: quad_corners = 76'd1563795843838783490619;
            12'd889: quad_corners = 76'd1563795843838783490619;
            12'd890: quad_corners = 76'd1563795843838783490619;
            12'd891: quad_corners = 76'd1563795843838783490619;
            12'd892: quad_corners = 76'd1563795843838783490619;
            12'd893: quad_corners = 76'd1563795843838783490619;
            12'd894: quad_corners = 76'd1563795843838783490619;
            12'd895: quad_corners = 76'd1563795843838783490619;
            12'd896: quad_corners = 76'd166903815503556664320;
            12'd897: quad_corners = 76'd166903815503556664320;
            12'd898: quad_corners = 76'd166903815503556664320;
            12'd899: quad_corners = 76'd166903815503556664320;
            12'd900: quad_corners = 76'd166903815503556664320;
            12'd901: quad_corners = 76'd166903815503556664320;
            12'd902: quad_corners = 76'd166903815503556664320;
            12'd903: quad_corners = 76'd166903815503556664320;
            12'd904: quad_corners = 76'd166903815503556664320;
            12'd905: quad_corners = 76'd166903815503556664320;
            12'd906: quad_corners = 76'd166903815503556664320;
            12’d907: quad_corners = 76’d166903815503556664320;
            12'd953: quad_corners = 76'd15637958438387834906l9;
```

```
                12'd999: quad_corners = 76'd38566250079808677480l;
                12'd1000: quad_corners = 76'd459304797854041896980;
                12'd1001: quad_corners = 76'd532802979996530634774;
                12'd1002: quad_corners = 76'd606445136314729490457;
                12'd1003: quad_corners = 76'd680087433370416177180;
                12'd1004: quad_corners = 76'd753729730151224956958;
                12'd1005: quad_corners = 76'd827227771281079520801;
                12'd1006: quad_corners = 76'd900870068061619865125;
                12'd1007: quad_corners = 76'd1048299200674119531560;
                12'd1008: quad_corners = 76'd1121941356717172044332;
                12'd1009: quad_corners = 76'd1195439397571879741487;
                12'd1010: quad_corners = 76'd1342868670646990380595;
                12'd1011: quad_corners = 76'd1490297802984343704631;
                12'd1012: quad_corners = 76'd1563795843838783490619;
                12'd1013: quad_corners = 76'd1563795843838783490619;
                12'd1014: quad_corners = 76'd1563795843838783490619;
                12'd1015: quad_corners = 76'd1563795843838783490619;
                12'd1016: quad_corners = 76'd1563795843838783490619;
                12'd1017: quad_corners = 76'd1563795843838783490619;
                12'd1018: quad_corners = 76'd1563795843838783490619;
                12'd1019: quad_corners = 76'd1563795843838783490619;
                12'd1020: quad_corners = 76'd1563795843838783490619;
                12'd1021: quad_corners = 76'd1563795843838783490619;
                12'd1022: quad_corners = 76'd1563795843838783490619;
                12'd1023: quad_corners = 76'd1563795843838783490619;
                12'd1024: quad_corners = 76'd166903815503556664320;
                12'd1025: quad_corners = 76'd166903815503556664320;
                12'd1026: quad_corners = 76'd166903815503556664320;
                12'd1027: quad_corners = 76'd166903815503556664320;
                12'd1028: quad_corners = 76'd166903815503556664320;
                12'd1029: quad_corners = 76'd166903815503556664320;
                12'd1030: quad_corners = 76'd166903815503556664320;
                12'd1031: quad_corners = 76'd166903815503556664320;
                12'd1032: quad_corners = 76'd166903815503556664320;
                12'd1033: quad_corners = 76'd166903815503556664320;
                12'd1034: quad_corners = 76'd166903815503556664320;
                12'd1035: quad_corners = 76'd166903815503556664320;
                12'd1036: quad_corners = 76'd166903815503556664320;
                12'd1037: quad_corners = 76'd166903815503556664320;
                12'd1038: quad_corners = 76'd166903815503556664320;
                12'd1039: quad_corners = 76'd166903815503556664320;
                12'd1040: quad_corners = 76'd166903815503556664320;
                12'd1041: quad_corners = 76'd166903815503556664320;
                12'd1042: quad_corners = 76'd166903815503556664320;
                12'd1043: quad_corners = 76'd166903815503556664320;
                12'd1044: quad_corners = 76'd166903815503556664320;
```

```
            12’d1045: quad_corners = 76’d16690381550355664320;
            12’d1046: quad_corners = 76’d16690381550355664320;
            12’d1047: quad_corners = 76’d16690381550355664320;
            12’d1048: quad_corners = 76’d16690381550355664320;
            12’d1049: quad_corners = 76’d16675913736552686131 2;
            12’d1050: quad_corners = 76’d9268350848207187609 6;
            12’d1051: quad_corners = 76’d9253883034404259737 6;
            12’d1052: quad_corners = 76’d9239429294350115020 8;
            12’d1053: quad_corners = 76’d9224961480547187148 9;
            12’d1054: quad_corners = 76’d9210493666744259277 0;
            12’d1055: quad_corners = 76’d9196039899202376243 5;
            12'd1091: quad_corners = 76'd17455098831505222144;
            12'd1092: quad_corners = 76'd17455098831505222144;
            12'd1093: quad_corners = 76'd17455098831505222144;
            12'd1094: quad_corners = 76'd17455098831505222144;
            12'd1095: quad_corners = 76'd17455098831505222144;
            12'd1096: quad_corners = 76'd17455098831505222144;
            12'd1097: quad_corners = 76'd17455098831505222144;
            12'd1098: quad_corners = 76'd17455098831505222144;
            12'd1099: quad_corners = 76'd17455098831505222144;
            12'd1100: quad_corners = 76'd17455098831505222144;
            12'd1101: quad_corners = 76'd17455098831505222144;
            12'd1102: quad_corners = 76'd17455098831505222144;
            12'd1103: quad_corners = 76'd17455098831505222144;
            12'd1104: quad_corners = 76'd17455098831505222144;
            12'd1105: quad_corners = 76'd17455098831505222144;
            12'd1106: quad_corners = 76'd17455098831505222144;
            12'd1107: quad_corners = 76'd17455098831505222144;
            12'd1108: quad_corners = 76'd17455098831505222144;
            12'd1109: quad_corners = 76'd17455098831505222144;
            12'd1110: quad_corners = 76'd17455098831505222144;
            12'd1111: quad_corners = 76'd17455098831505222144;
            12'd1112: quad_corners = 76'd17455098831505222144;
            12'd1113: quad_corners = 76'd17310420693475419648;
            12'd1114: quad_corners = 76'd17165883292934496768;
            12'd1115: quad_corners = 76'd17021205154905218048;
            12'd1116: quad_corners = 76'd16876667754364295168;
            12'd1117: quad_corners = 76'd16731989616334492160;
            12'd1118: quad_corners = 76'd16731567403869424640;
            12'd1119: quad_corners = 76'd16586889265840146432;
            12'd1120: quad_corners = 76'd16442351590421316098;
            12'd1121: quad_corners = 76'd16297672901830917123;
            12'd1122: quad_corners = 76'd8993997023320081766 9;
            12'd1123: quad_corners = 76'd8979529126946407578 3;
            12'd1124: quad_corners = 76'd163581844526129752073;
            12'd1125: quad_corners = 76'd163437165562393010699;
            12'd1126: quad_corners = 76'd237079462893494475277;
            12'd1127: quad_corners = 76'd310721759949718033423;
            12'd1128: quad_corners = 76'd384508172468626918418;
            12'd1129: quad_corners = 76'd458150469524582040597;
            12'd1130: quad_corners = 76'd531792766855146634264;
            12'd1131: quad_corners = 76'd605579179099178136603;
            12'd1132: quad_corners = 76'd679221476154864823326;
            12'd1133: quad_corners = 76'd752863773210283074593;
            12'd1134: quad_corners = 76'd900437021011395992612;
            12'd1135: quad_corners = 76'd974079318066814243880;
            12'd1136: quad_corners = 76'd1121508591417071225388;
            12'd1137: quad_corners = 76'd1195294862648199070256;
            12'd1138: quad_corners = 76'd1342724135998187616308;
            12'd1139: quad_corners = 76'd1490153409073566166584;
            12'd1140: quad_corners = 76'd1637726656874142213180;
            12'd1141: quad_corners = 76'd1637726656874142213180;
            12'd1142: quad_corners = 76'd1637726656874142213180;
            12'd1143: quad_corners = 76'd1637726656874142213180;
            12'd1144: quad_corners = 76'd1637726656874142213180;
            12'd1145: quad_corners = 76'd1637726656874142213180;
            12'd1146: quad_corners = 76'd1637726656874142213180;
            12'd1147: quad_corners = 76'd1637726656874142213180;
            12'd1148: quad_corners = 76'd1637726656874142213180;
            12'd1149: quad_corners = 76'd1637726656874142213180;
            12'd1150: quad_corners = 76'd1637726656874142213180;
            12'd1151: quad_corners = 76'd1637726656874142213180;
            12'd1152: quad_corners = 76'd15724449937206048768;
            12'd1153: quad_corners = 76'd15724449937206048768;
            12'd1154: quad_corners = 76'd15724449937206048768;
            12'd1155: quad_corners = 76'd15724449937206048768;
            12'd1156: quad_corners = 76'd15724449937206048768;
            12'd1157: quad_corners = 76'd15724449937206048768;
            12'd1158: quad_corners = 76'd15724449937206048768;
            12'd1159: quad_corners = 76'd15724449937206048768;
            12'd1160: quad_corners = 76'd15724449937206048768;
            12'd1161: quad_corners = 76'd15724449937206048768;
            12'd1162: quad_corners = 76'd15724449937206048768;
            12'd1163: quad_corners = 76'd15724449937206048768;
            12'd1164: quad_corners = 76'd15724449937206048768;
            12'd1165: quad_corners = 76'd15724449937206048768;
            12'd1166: quad_corners = 76'd15724449937206048768;
            12'd1167: quad_corners = 76'd15724449937206048768;
            12'd1168: quad_corners = 76'd15724449937206048768;
            12'd1169: quad_corners = 76'd15724449937206048768;
            12'd1170: quad_corners = 76'd15724449937206048768;
            12'd1171: quad_corners = 76'd15724449937206048768;
            12'd1172: quad_corners = 76'd15724449937206048768;
            12'd1173: quad_corners = 76'd15724449937206048768;
            12'd1174: quad_corners = 76'd15724449937206048768;
            12'd1175: quad_corners = 76'd15724449937206048768;
            12'd1176: quad_corners = 76'd15724449937206048768;
            12'd1177: quad_corners = 76'd15579771799176246272;
            12'd1178: quad_corners = 76'd15435234398635323392;
            12'd1179: quad_corners = 76'd15434812186169732096;
            12'd1180: quad_corners = 76'd15290134048140453376;
            12'd1181: quad_corners = 76'd15145596647599530496;
            12'd1182: quad_corners = 76'd15145174435134462976;
```

```
12’d1183: quad_corners = 76’d15000496297105184768;
12’d1184: quad_corners = 76’d15000074084639592960;
12’d1185: quad_corners = 76’d14855395946341878785;
12’d1186: quad_corners = 76’d14854972908437784067;
12’d1187: quad_corners = 76’d14710293944969478149;
12’d1188: quad_corners = 76’d14709871181943290375;
12’d1189: quad_corners = 76’d88496283701120611337;
12’d1190: quad_corners = 76’d162138581032222075915;
12’d1191: quad_corners = 76’d235925134288887752206;
12’d1192: quad_corners = 76’d309711546808065073169;
12’d1193: quad_corners = 76’d383497959326974482452;
12’d1194: quad_corners = 76’d457140397120417959959;
12’d1195: quad_corners = 76’d530926809639326845466;
12’d1196: quad_corners = 76’d678500198453073936925;
12’d1197: quad_corners = 76’d752286610696837003809;
12’d1198: quad_corners = 76’d826073022940867981860;
12’d1199: quad_corners = 76’d973646411754078202408;
12’d1200: quad_corners = 76’d1121219941030167831084;
12’d1201: quad_corners = 76’d1195006353273661937712;
12’d1202: quad_corners = 76’d1342579741812263211061;
12’d1203: quad_corners = 76’d1490153130350327088697;
12’d1204: quad_corners = 76’d1637726518613782019646;
12’d1205: quad_corners = 76’d1637726518613782019646;
12’d1206: quad_corners = 76’d1637726518613782019646;
12’d1207: quad_corners = 76’d1637726518613782019646;
12’d1208: quad_corners = 76’d1637726518613782019646;
12’d1209: quad_corners = 76’d1637726518613782019646;
12’d1210: quad_corners = 76’d1637726518613782019646;
12’d1211: quad_corners = 76’d1637726518613782019646;
12’d1212: quad_corners = 76’d1637726518613782019646;
12’d1213: quad_corners = 76’d1637726518613782019646;
12’d1214: quad_corners = 76’d1637726518613782019646;
12’d1215: quad_corners = 76’d1637726518613782019646;
12’d1216: quad_corners = 76’d13993801042906351104;
12’d1217: quad_corners = 76’d13993801042906351104;
12’d1218: quad_corners = 76’d13993801042906351104;
12’d1219: quad_corners = 76’d13993801042906351104;
12’d1220: quad_corners = 76’d13993801042906351104;
12’d1221: quad_corners = 76’d13993801042906351104;
12’d1222: quad_corners = 76’d13993801042906351104;
12’d1223: quad_corners = 76’d13993801042906351104;
12’d1224: quad_corners = 76’d13993801042906351104;
12’d1225: quad_corners = 76’d13993801042906351104;
12’d1226: quad_corners = 76’d13993801042906351104;
12’d1227: quad_corners = 76’d13993801042906351104;
12’d1228: quad_corners = 76’d13993801042906351104;
```

```
                12’d1229: quad_corners = 76’d13993801042906351104;
                12’d1230: quad_corners = 76’d13993801042906351104;
                12’d1231: quad_corners = 76’d13993801042906351104;
                12’d1232: quad_corners = 76’d13993801042906351104;
                12’d1233: quad_corners = 76’d13993801042906351104;
                12’d1234: quad_corners = 76’d13993801042906351104;
                12’d1235: quad_corners = 76’d13993801042906351104;
                12’d1236: quad_corners = 76’d13993801042906351104;
                12’d1237: quad_corners = 76’d13993801042906351104;
                12’d1238: quad_corners = 76’d13993801042906351104;
                12’d1239: quad_corners = 76’d13993801042906351104;
                12’d1240: quad_corners = 76’d13993801042906351104;
                12’d1241: quad_corners = 76’d13993378830440759808;
                12’d1242: quad_corners = 76’d13848841429899836928;
                12’d1243: quad_corners = 76’d13848419217434769920;
                12’d1244: quad_corners = 76’d13703741079404967424;
                12’d1245: quad_corners = 76’d13703318866939899904;
                12’d1246: quad_corners = 76’d13702896654474832896;
                12’d1247: quad_corners = 76’d13558359253933909504;
                12’d1248: quad_corners = 76’d13557796303979962880;
                12’d1249: quad_corners = 76’d13557374091514895360;
                12’d1250: quad_corners = 76’d13556951878512956929;
                12’d1251: quad_corners = 76’d13556388378266849283;
                12’d1252: quad_corners = 76’d13555965340362754565;
                12’d1253: quad_corners = 76’d13555542577336566791;
                12’d1254: quad_corners = 76’d87341955096513887754;
                12’d1255: quad_corners = 76’d161128508353447999501;
                12’d1256: quad_corners = 76’d234914920872625320464;
                12’d1257: quad_corners = 76’d308845589592245283347;
                12’d1258: quad_corners = 76’d456418978405992375318;
                12’d1259: quad_corners = 76’d530205531662658051097;
                12’d1260: quad_corners = 76’d603992084919055291933;
                12’d1261: quad_corners = 76’d751709588646000332320;
                12’d1262: quad_corners = 76’d825496141902398096932;
                12’d1263: quad_corners = 76’d973213645903952609320;
                12’d1264: quad_corners = 76’d1120787175180042237997;
                12’d1265: quad_corners = 76’d1268504679181596749873;
                12’d1266: quad_corners = 76’d1416222182908273354293;
                12’d1267: quad_corners = 76’d1563795712184094547514;
                12’d1268: quad_corners = 76’d1711513215910771152447;
                12’d1269: quad_corners = 76’d1711513215910771152447;
                12’d1270: quad_corners = 76’d1711513215910771152447;
                12’d1271: quad_corners = 76’d1711513215910771152447;
                12’d1272: quad_corners = 76’d1711513215910771152447;
                12’d1273: quad_corners = 76’d1711513215910771152447;
                12’d1274: quad_corners = 76’d1711513215910771152447;
```

```
            12'd1275: quad_corners = 76'd17115132159107711524447;
            12'd1276: quad_corners = 76'd17115132159107711524447;
            12'd1277: quad_corners = 76'd17115132159107711524447;
            12'd1278: quad_corners = 76'd17115132159107711524447;
            12'd1279: quad_corners = 76'd17115132159107711524447;
            12'd1280: quad_corners = 76'd1240740807417034035 2;
            12'd1281: quad_corners = 76'd12407408074170340352;
            12'd1282: quad_corners = 76'd12407408074170340352;
            12'd1283: quad_corners = 76'd12407408074170340352;
            12'd1284: quad_corners = 76'd12407408074170340352;
            12'd1285: quad_corners = 76'd12407408074170340352;
            12'd1286: quad_corners = 76'd12407408074170340352;
            12'd1287: quad_corners = 76'd12407408074170340352;
            12'd1288: quad_corners = 76'd12407408074170340352;
            12'd1289: quad_corners = 76'd12407408074170340352;
            12'd1290: quad_corners = 76'd12407408074170340352;
            12'd1291: quad_corners = 76'd12407408074170340352;
            12'd1292: quad_corners = 76'd12407408074170340352;
            12'd1293: quad_corners = 76'd12407408074170340352;
            12'd1294: quad_corners = 76'd12407408074170340352;
            12'd1295: quad_corners = 76'd12407408074170340352;
            12'd1296: quad_corners = 76'd12407408074170340352;
            12'd1297: quad_corners = 76'd12407408074170340352;
            12'd1298: quad_corners = 76'd12407408074170340352;
            12'd1299: quad_corners = 76'd12407408074170340352;
            12'd1300: quad_corners = 76'd12407408074170340352;
            12'd1301: quad_corners = 76'd12407408074170340352;
            12'd1302: quad_corners = 76'd12407408074170340352;
            12'd1303: quad_corners = 76'd12407408074170340352;
            12'd1304: quad_corners = 76'd12407408074170340352;
            12'd1305: quad_corners = 76'd12406985861705273344;
            12'd1306: quad_corners = 76'd12262448461164350464;
            12'd1307: quad_corners = 76'd12262026248698759168;
            12'd1308: quad_corners = 76'd12261604036233691648;
            12'd1309: quad_corners = 76'd12261041086280269312;
            12'd1310: quad_corners = 76'd12260618873814678016;
            12'd1311: quad_corners = 76'd12260196661349610496;
            12'd1312: quad_corners = 76'd12259774448884543488;
            12'd1313: quad_corners = 76'd12259352236419475968;
            12'd1314: quad_corners = 76'd12258930023954408960;
            12'd1315: quad_corners = 76'd12402622999028326401;
            12'd1316: quad_corners = 76'd12402200236002138628;
            12'd1317: quad_corners = 76'd12401636735756030982;
            12'd1318: quad_corners = 76'd8633230518076599 9113;
            12'd1319: quad_corners = 76'd160118858712309582348;
            12'd1320: quad_corners = 76'd234049527157319549455;
            12'd1321: quad_corners = 76'd3079801958769389885 62;
            12'd1367: quad_corners = 76'd10821015105434329088;
            12'd1368: quad_corners = 76'd10821015105434329088;
            12'd1369: quad_corners = 76'd10820592892969262080;
            12'd1370: quad_corners = 76'd10820170680504195072;
            12'd1371: quad_corners = 76'd10819748468038603776;
            12'd1372: quad_corners = 76'd10819326255573536768;
            12'd1373: quad_corners = 76'd10818904043108469760;
            12'd1374: quad_corners = 76'd10962597018718734336;
            12'd1375: quad_corners = 76'd10962174806253666816;
            12'd1376: quad_corners = 76'd10961752593788599808;
            12'd1377: quad_corners = 76'd11105445569399388672;
            12'd1378: quad_corners = 76'd11105023356934321152;
            12'd1379: quad_corners = 76'd11248716332545110016;
            12'd1380: quad_corners = 76'd11248294119274736131;
            12'd1381: quad_corners = 76'd11392127282081195013;
            12'd1382: quad_corners = 76'd11535819707399299080;
            12'd1383: quad_corners = 76'd85466488427287173643;
            12'd1384: quad_corners = 76'd233184133442013254670;
            12'd1385: quad_corners = 76'd307114802161901129234;
            12'd1386: quad_corners = 76'd381045470606642661397;
            12'd1387: quad_corners = 76'd528763115621368742425;
            12'd1388: quad_corners = 76'd602837899529064037405;
            12'd1389: quad_corners = 76'd750555544268644299809;
            12'd1390: quad_corners = 76'd898273329745712393765;
            12'd1391: quad_corners = 76'd1046135089947977459753;
            12'd1392: quad_corners = 76'd1193852734687557722158;
            12'd1393: quad_corners = 76'd1341714494614944881203;
            12'd1394: quad_corners = 76'd1489576254817210470968;
            12'd1395: quad_corners = 76'd1637438014744329194045;
            12'd1396: quad_corners = 76'd1785299915409204708418;
            12'd1397: quad_corners = 76'd1785299915409204708418;
            12'd1398: quad_corners = 76'd1785299915409204708418;
            12'd1399: quad_corners = 76'd1785299915409204708418;
            12'd1400: quad_corners = 76'd1785299915409204708418;
            12'd1401: quad_corners = 76'd1785299915409204708418;
            12'd1402: quad_corners = 76'd1785299915409204708418;
            12'd1403: quad_corners = 76'd1785299915409204708418;
            12'd1404: quad_corners = 76'd1785299915409204708418;
            12'd1405: quad_corners = 76'd1785299915409204708418;
            12'd1406: quad_corners = 76'd1785299915409204708418;
            12'd1407: quad_corners = 76'd1785299915409204708418;
            12'd1408: quad_corners = 76'd95229932503383848896;
            12'd1409: quad_corners = 76'd95229932503383848896;
            12'd1410: quad_corners = 76'd95229932503383848896;
            12'd1411: quad_corners = 76'd95229932503383848896;
            12'd1412: quad_corners = 76'd95229932503383848896;
            12'd1413: quad_corners = 76'd9522993250338384896;
            12'd1414: quad_corners = 76'd9522993250338384896;
            12'd1415: quad_corners = 76'd9522993250338384896;
            12'd1416: quad_corners = 76'd9522993250338384896;
            12'd1417: quad_corners = 76'd9522993250338384896;
            12'd1418: quad_corners = 76'd9522993250338384896;
            12'd1419: quad_corners = 76'd9522993250338384896;
            12'd1420: quad_corners = 76'd9522993250338384896;
            12'd1421: quad_corners = 76'd9522993250338384896;
            12'd1422: quad_corners = 76'd9522993250338384896;
            12'd1423: quad_corners = 76'd9522993250338384896;
            12'd1424: quad_corners = 76'd9522993250338384896;
            12'd1425: quad_corners = 76'd9522993250338384896;
            12'd1426: quad_corners = 76'd9522993250338384896;
            12'd1427: quad_corners = 76'd9522993250338384896;
            12'd1428: quad_corners = 76'd9522993250338384896;
            12'd1429: quad_corners = 76'd9522993250338384896;
            12'd1430: quad_corners = 76'd9522993250338384896;
            12'd1431: quad_corners = 76'd9522993250338384896;
            12'd1432: quad_corners = 76'd9522993250338384896;
            12'd1433: quad_corners = 76'd9522571037873317888;
            12'd1434: quad_corners = 76'd9522148825407726592;
            12'd1435: quad_corners = 76'd9521726612942659584;
            12'd1436: quad_corners = 76'd9521304400477068288;
            12'd1437: quad_corners = 76'd9520882188012001280;
            12'd1438: quad_corners = 76'd9664575163622790144;
            12'd1439: quad_corners = 76'd9664293688646078464;
            12'd1440: quad_corners = 76'd9807986664256343040;
            12'd1441: quad_corners = 76'd9951679639867131904;
            12'd1442: quad_corners = 76'd9951257427402064384;
            12'd1443: quad_corners = 76'd10094950403012853248;
            12'd1444: quad_corners = 76'd10238784115843561474;
            12'd1445: quad_corners = 76'd10382477090917479429;
            12'd1446: quad_corners = 76'd10526169791113489928;
            12'd1447: quad_corners = 76'd8460109443683401 1659;
            12'd1448: quad_corners = 76'd23231873945156009268 6;
            12'd1449: quad_corners = 76'd30624940817171640270 6;
            12'd1450: quad_corners = 76'd38032433281716848846 9;
            12'd1451: quad_corners = 76'd52818609301997094965 7;
            12'd1452: quad_corners = 76'd67590387877191695055 7;
            12'd1453: quad_corners = 76'd74997866267988068099 3;
            12'd1454: quad_corners = 76'd89784042260753627549 4;
            12'd1455: quad_corners = 76'd104570232354755865655 4;
            12'd1456: quad_corners = 76'd119356408375009215799 9;
            12'd1457: quad_corners = 76'd134142598441523610782 8;
            12'd1458: quad_corners = 76'd148928774461750169759 3;
            12'd1459: quad_corners = 76'd17110807367652912743O2;
            12'd1505: quad_corners = 76'd8797913710334874624;
            12'd1506: quad_corners = 76'd8941606685945663488;
            12'd1507: quad_corners = 76'd9085440399044807680;
            12'd1508: quad_corners = 76'd9229133374655596034;
            12'd1509: quad_corners = 76'd9517082275562160645;
            12'd1510: quad_corners = 76'd9660775250904514056;
            12'd1511: quad_corners = 76'd15752267646634114868З;
            12'd1512: quad_corners = 76'd23145334546110693019О;
            12'd1513: quad_corners = 76'd30552827010682745192;
            12'd1514: quad_corners = 76'd45339017104738617960;
            12'd1515: quad_corners = 76'd52746495495535043484;
            12'd1516: quad_corners = 76'd67532685617051863401;
            12'd1517: quad_corners = 76'd82318875711080892624;
            12'd1518: quad_corners = 76'd97105051731361086314;
            12'd1519: quad_corners = 76'd111905653344170910007;
            12'd1520: quad_corners = 76'd126691843438199939230;
            12'd1521: quad_corners = 76'd141492445051009710494;
            12'd1522: quad_corners = 76'd156293046636331743493;
            12'd1523: quad_corners = 76'd178457934359817749817;
            12'd1524: quad_corners = 76'd193258535972600729965;
            12'd1525: quad_corners = 76'd193258535972600729965;
            12'd1526: quad_corners = 76'd193258535972600729965;
            12'd1527: quad_corners = 76'd193258535972600729965;
            12'd1528: quad_corners = 76'd193258535972600729965;
            12'd1529: quad_corners = 76'd193258535972600729965;
            12'd1530: quad_corners = 76'd193258535972600729965;
            12'd1531: quad_corners = 76'd193258535972600729965;
            12'd1532: quad_corners = 76'd193258535972600729965;
            12'd1533: quad_corners = 76'd193258535972600729965;
            12'd1534: quad_corners = 76'd193258535972600729965;
            12'd1535: quad_corners = 76'd193258535972600729965;
            12'd1536: quad_corners = 76'd6782834352069067264;
            12'd1537: quad_corners = 76'd6782834352069067264;
            12'd1538: quad_corners = 76'd6782834352069067264;
            12'd1539: quad_corners = 76'd6782834352069067264;
            12'd1540: quad_corners = 76'd6782834352069067264;
            12'd1541: quad_corners = 76'd6782834352069067264;
            12'd1542: quad_corners = 76'd6782834352069067264;
            12'd1543: quad_corners = 76'd6782834352069067264;
            12'd1544: quad_corners = 76'd6782834352069067264;
            12'd1545: quad_corners = 76'd6782834352069067264;
            12'd1546: quad_corners = 76'd6782834352069067264;
            12'd1547: quad_corners = 76'd6782834352069067264;
            12'd1548: quad_corners = 76'd6782834352069067264;
            12'd1549: quad_corners = 76'd6782834352069067264;
            12'd1550: quad_corners = 76'd6782834352069067264;
```

```
                    12'd1551: quad_corners = 76'd6782834352069067264;
                    12'd1552: quad_corners = 76'd6782834352069067264;
                    12'd1553: quad_corners = 76'd6782834352069067264;
                    12'd1554: quad_corners = 76'd6782834352069067264;
                    12'd1555: quad_corners = 76'd6782834352069067264;
                    12'd1556: quad_corners = 76'd6782834352069067264;
                    12'd1557: quad_corners = 76'd6782834352069067264;
                    12'd1558: quad_corners = 76'd6782834352069067264;
                    12'd1559: quad_corners = 76'd6782834352069067264;
                    12'd1560: quad_corners = 76'd6782834352069067264;
                    12'd1561: quad_corners = 76'd6926527327679856128;
                    12'd1562: quad_corners = 76'd6926105115214789120;
                    12'd1563: quad_corners = 76'd6925682902749198336;
                    12'd1564: quad_corners = 76'd7069516615848342528;
                    12'd1565: quad_corners = 76'd7213209591459131392;
                    12'd1566: quad_corners = 76'd7212928116481895424;
                    12'd1567: quad_corners = 76'd7356621092092684288;
                    12'd1568: quad_corners = 76'd7500454805191828480;
                    12'd1569: quad_corners = 76'd7788262968878473216;
                    12'd1570: quad_corners = 76'd7932096681977617408;
                    12'd1571: quad_corners = 76'd8075789657588406272;
                    12'd1572: quad_corners = 76'd8363738558763406338;
                    12'd1573: quad_corners = 76'd8507572271862550533;
                    12'd1574: quad_corners = 76'd82582497467607321096;
                    12'd1575: quad_corners = 76'd156657281925863736843;
                    12'd1576: quad_corners = 76'd230732207121340072463;
                    12'd1577: quad_corners = 76'd378594108061898800146;
                    12'd1578: quad_corners = 76'd452669032982497228822;
                    12'd1579: quad_corners = 76'd600675049111131812378;
                    12'd1580: quad_corners = 76'd748536950326300012062;
                    12'd1581: quad_corners = 76'd896398851266859264035;
                    12'd1582: quad_corners = 76'd1044404867395225412135;
                    12'd1583: quad_corners = 76'd1192410883798469467692;
                    12'd1584: quad_corners = 76'd1340416899926835615793;
                    12'd1585: quad_corners = 76'd1488422916054933852726;
                    12'd1586: quad_corners = 76'd1636428932183300000827;
                    12'd1587: quad_corners = 76'd1858221924881114351169;
                    12'd1588: quad_corners = 76'd2006228081746700419143;
                    12'd1589: quad_corners = 76'd2006228081746700419143;
                    12'd1590: quad_corners = 76'd2006228081746700419143;
                    12'd1591: quad_corners = 76'd2006228081746700419143;
                    12'd1592: quad_corners = 76'd2006228081746700419143;
                    12'd1593: quad_corners = 76'd2006228081746700419143;
                    12'd1594: quad_corners = 76'd2006228081746700419143;
                    12'd1595: quad_corners = 76'd2006228081746700419143;
                    12'd1596: quad_corners = 76'd2006228081746700419143;
```

```
        12’d1597: quad_corners = 76’d200622808174670041914 3;
```

```
            12'd1643: quad_corners = 76'd673740887341080543771;
            12'd1689: quad_corners = 76'd4618854731125412352;
            12'd1690: quad_corners = 76'd4618573256148700672;
            12'd1691: quad_corners = 76'd4762266231759490048;
            12'd1692: quad_corners = 76'd4906099944858109952;
            12'd1693: quad_corners = 76'd5049933657957254144;
            12'd1694: quad_corners = 76'd5193626633568043520;
            12'd1695: quad_corners = 76'd5337460346667187712;
            12'd1696: quad_corners = 76'd5625409247841663488;
            12'd1697: quad_corners = 76'd5769242960940808192;
            12'd1698: quad_corners = 76'd6057191862115808256;
            12'd1699: quad_corners = 76'd6345140763290808320;
            12'd1700: quad_corners = 76'd6633089664465808387;
            12'd1701: quad_corners = 76'd80708014860479014918;
            12'd1702: quad_corners = 76'd154782940056492221449;
            12'd1703: quad_corners = 76'd302644841547343634445;
            12'd1704: quad_corners = 76'd376863881931432696848;
            12'd1705: quad_corners = 76'd524725924159504029716;
            12'd1706: quad_corners = 76'd598944964543056221208;
            12'd1707: quad_corners = 76'd746950980946837671964;
            12'd1708: quad_corners = 76'd894957138087838517792;
            12'd1709: quad_corners = 76'd1042963154491351008805;
            12'd1710: quad_corners = 76'd1191113426820427711018;
            12'd1711: quad_corners = 76'd1339119443223940725807;
            12'd1712: quad_corners = 76'd1487269715553017428020;
            12'd1713: quad_corners = 76'd1709062848988588045369;
            12'd1714: quad_corners = 76'd1857213121317665271358;
            12'd1715: quad_corners = 76'd2005363252634107275844;
            12'd1716: quad_corners = 76'd2227300501257754272842;
            12'd1717: quad_corners = 76'd2227300501257754272842;
            12'd1718: quad_corners = 76'd2227300501257754272842;
            12'd1719: quad_corners = 76'd2227300501257754272842;
            12'd1720: quad_corners = 76'd2227300501257754272842;
            12'd1721: quad_corners = 76'd2227300501257754272842;
            12'd1722: quad_corners = 76'd2227300501257754272842;
            12'd1723: quad_corners = 76'd2227300501257754272842;
            12'd1724: quad_corners = 76'd2227300501257754272842;
            12'd1725: quad_corners = 76'd2227300501257754272842;
            12'd1726: quad_corners = 76'd2227300501257754272842;
            12'd1727: quad_corners = 76'd2227300501257754272842;
            12'd1728: quad_corners = 76'd3465511012983430144;
            12'd1729: quad_corners = 76'd3465511012983430144;
            12'd1730: quad_corners = 76'd3465511012983430144;
            12'd1731: quad_corners = 76'd3465511012983430144;
            12'd1732: quad_corners = 76'd3465511012983430144;
            12'd1733: quad_corners = 76'd3465511012983430144;
            12'd1734: quad_corners = 76'd3465511012983430144;
            12'd1735: quad_corners = 76'd3465511012983430144;
            12'd1736: quad_corners = 76'd3465511012983430144;
            12'd1737: quad_corners = 76'd3465511012983430144;
            12'd1738: quad_corners = 76'd3465511012983430144;
            12'd1739: quad_corners = 76'd3465511012983430144;
            12'd1740: quad_corners = 76'd3465511012983430144;
            12'd1741: quad_corners = 76'd3465511012983430144;
            12'd1742: quad_corners = 76'd3465511012983430144;
            12'd1743: quad_corners = 76'd3465511012983430144;
            12'd1744: quad_corners = 76'd3465511012983430144;
            12'd1745: quad_corners = 76'd3465511012983430144;
            12'd1746: quad_corners = 76'd3465511012983430144;
            12'd1747: quad_corners = 76'd3465511012983430144;
            12'd1748: quad_corners = 76'd3465511012983430144;
            12'd1749: quad_corners = 76'd3465511012983430144;
            12'd1750: quad_corners = 76'd3465511012983430144;
            12'd1751: quad_corners = 76'd3465511012983430144;
            12'd1752: quad_corners = 76'd3465511012983430144;
            12'd1753: quad_corners = 76'd3609344726351010304;
            12'd1754: quad_corners = 76'd3608922514154379264;
            12'd1755: quad_corners = 76'd3752756227789870080;
            12'd1756: quad_corners = 76'd3896589940889014784;
            12'd1757: quad_corners = 76'd4040282916499803648;
            12'd1758: quad_corners = 76'd4328231817674804224;
            12'd1759: quad_corners = 76'd4472065530773424128;
            12'd1760: quad_corners = 76'd4760014431948424192;
            12'd1761: quad_corners = 76'd4903848145047568896;
            12'd1762: quad_corners = 76'd5191797046222568960;
            12'd1763: quad_corners = 76'd5479745947397569026;
            12'd1764: quad_corners = 76'd79554671143410776068;
            12'd1765: quad_corners = 76'd227416713371750544391;
            12'd1766: quad_corners = 76'd301635753755839606795;
            12'd1767: quad_corners = 76'd375710678951852813326;
            12'd1768: quad_corners = 76'd449929860073430231058;
            12'd1769: quad_corners = 76'd597935876752357499925;
            12'd1770: quad_corners = 76'd745942034168505212953;
            12'd1771: quad_corners = 76'd820161074552325839901;
            12'd1772: quad_corners = 76'd968167231968473028642;
            12'd1773: quad_corners = 76'd1116317504297549730854;
            12'd1774: quad_corners = 76'd1264323661438819536939;
            12'd1775: quad_corners = 76'd1412473933767896239152;
            12'd1776: quad_corners = 76'd1634411182391811147317;
            12'd1777: quad_corners = 76'd1782561454995766280763;
            12'd1778: quad_corners = 76'd1930711727324842982976;
            12'd1779: quad_corners = 76'd2152648975948489455686;
            12'd1780: quad_corners = 76'd2374730339760480744524;
            12’d1781: quad_corners = 76’d237473033976048074452 4;
            12'd1827: quad_corners = 76'd152332559646744953859;
            12'd1873: quad_corners = 76'd1734721379585289216;
            12’d1919: quad_corners = 76’d2595514668532015862351;
            12’d1920: quad_corners = 76’d746563028577239004l6;
            12’d1921: quad_corners = 76’d74656302857723900416;
            12’d1922: quad_corners = 76’d74656302857723900416;
            12’d1923: quad_corners = 76’d74656302857723900416;
            12’d1924: quad_corners = 76’d74656302857723900416;
            12’d1925: quad_corners = 76’d74656302857723900416;
            12’d1926: quad_corners = 76’d74656302857723900416;
            12’d1927: quad_corners = 76’d74656302857723900416;
            12’d1928: quad_corners = 76’d74656302857723900416;
            12’d1929: quad_corners = 76’d74656302857723900416;
            12’d1930: quad_corners = 76’d74656302857723900416;
            12’d1931: quad_corners = 76’d74656302857723900416;
            12’d1932: quad_corners = 76’d74656302857723900416;
            12’d1933: quad_corners = 76’d74656302857723900416;
            12’d1934: quad_corners = 76’d74656302857723900416;
            12’d1935: quad_corners = 76’d74656302857723900416;
            12’d1936: quad_corners = 76’d74656302857723900416;
            12’d1937: quad_corners = 76’d74656302857723900416;
            12’d1938: quad_corners = 76’d74656302857723900416;
            12’d1939: quad_corners = 76’d74656302857723900416;
            12’d1940: quad_corners = 76’d74656302857723900416;
            12’d1941: quad_corners = 76’d74656302857723900416;
            12’d1942: quad_corners = 76’d74656302857723900416;
            12’d1943: quad_corners = 76’d74656302857723900416;
            12’d1944: quad_corners = 76’d74656302857723900416;
            12’d1945: quad_corners = 76’d74800136571359916032;
            12’d1946: quad_corners = 76’d74943829547507576320;
            12’d1947: quad_corners = 76’d75087663260875156480;
            12’d1948: quad_corners = 76’d75231496974510647296;
            12’d1949: quad_corners = 76’d149306422170792289792;
            12’d1950: quad_corners = 76’d149450396621648225280;
            12’d1951: quad_corners = 76’d223525321818198303232;
            12’d1952: quad_corners = 76’d223813270719641739264;
            12’d1953: quad_corners = 76’d297888336653143301122;
            12’d1954: quad_corners = 76’d371963261849424943620;
            12’d1955: quad_corners = 76’d446038327782926505991;
            12’d1956: quad_corners = 76’d520257368167015568394;
            12’d1957: quad_corners = 76’d594332434100517130765;
            12’d1958: quad_corners = 76’d668551615222094548496;
            12’d1959: quad_corners = 76’d742770796343671966739;
            12’d1960: quad_corners = 76’d890776953760087590935;
            12’d1961: quad_corners = 76’d965140250069741389339;
            12’d1962: quad_corners = 76’d1113146407486157013535;
            12’d1963: quad_corners = 76’d1261296680090648493603;
            12’d1964: quad_corners = 76’d1335660117137790123047;
            12’d1965: quad_corners = 76’d14838103897422821274O4;
            12'd2011: quad_corners = 76'd22194047695941566l056;
```

```
                12'd2057: quad_corners = 76'd4427252262893123655 68;
                12'd2058: quad_corners = 76'd442725226289312365568;
                12'd2059: quad_corners = 76'd442725226289312365568;
                12'd2060: quad_corners = 76'd442725226289312365568;
                12'd2061: quad_corners = 76'd442725226289312365568;
                12'd2062: quad_corners = 76'd442725226289312365568;
                12'd2063: quad_corners = 76'd442725226289312365568;
                12'd2064: quad_corners = 76'd442725226289312365568;
                12'd2065: quad_corners = 76'd442725226289312365568;
                12'd2066: quad_corners = 76'd442725226289312365568;
                12'd2067: quad_corners = 76'd442725226289312365568;
                12'd2068: quad_corners = 76'd442725226289312365568;
                12'd2069: quad_corners = 76'd442725226289312365568;
                12'd2070: quad_corners = 76'd442725226289312365568;
                12'd2071: quad_corners = 76'd442725226289312365568;
                12'd2072: quad_corners = 76'd442725226289312365568;
                12'd2073: quad_corners = 76'd442724944539994618880;
                12'd2074: quad_corners = 76'd442724663065554778624;
                12'd2075: quad_corners = 76'd442724381591114938368;
                12'd2076: quad_corners = 76'd442868356042238785024;
                12'd2077: quad_corners = 76'd516943281238788862976;
                12'd2078: quad_corners = 76'd517087114952156443138;
                12'd2079: quad_corners = 76'd517375204591356669955;
                12'd2080: quad_corners = 76'd591450129787638312453;
                12'd2081: quad_corners = 76'd665525195721408309768;
                12'd2082: quad_corners = 76'd739744376843522598922;
                12'd2083: quad_corners = 76'd740032466482454390285;
                12'd2084: quad_corners = 76'd814251506866543977487;
                12'd2085: quad_corners = 76'd962257805020716392978;
                12'd2086: quad_corners = 76'd1036476986142562246677;
                12'd2087: quad_corners = 76'd1110696167539017571353;
                12'd2088: quad_corners = 76'd1184915348660594989596;
                12'd2089: quad_corners = 76'd1333065762002574825504;
                12'd2090: quad_corners = 76'd1407284943124152767524;
                12'd2091: quad_corners = 76'd1555435356466132603432;
                12'd2092: quad_corners = 76'd1703585769808112439340;
                12'd2093: quad_corners = 76'd1851736183150092799024;
                12'd2094: quad_corners = 76'd2000030711955026397749;
                12'd2095: quad_corners = 76'd2148181125297006233146;
                12'd2096: quad_corners = 76'd2296475653827062449215;
                12'd2097: quad_corners = 76'd2444770182357118140484;
                12'd2098: quad_corners = 76'd2666851687182012038729;
                12'd2099: quad_corners = 76'd2815146215711799294543;
                12'd2100: quad_corners = 76'd3037227720536424756821;
                12'd2101: quad_corners = 76'd3037227720536424756821;
                12'd2102: quad_corners = 76'd3037227720536424756821;
```

```
            12'd2103: quad_corners = 76'd3037227720536424756821;
            12'd2104: quad_corners = 76'd3037227720536424756821;
            12'd2105: quad_corners = 76'd3037227720536424756821;
            12'd2106: quad_corners = 76'd3037227720536424756821;
            12'd2107: quad_corners = 76'd3037227720536424756821;
            12'd2108: quad_corners = 76'd3037227720536424756821;
            12'd2109: quad_corners = 76'd3037227720536424756821;
            12'd2110: quad_corners = 76'd3037227720536424756821;
            12'd2111: quad_corners = 76'd3037227720536424756821;
            12'd2112: quad_corners = 76'd664085591123558202880;
            12'd2113: quad_corners = 76'd664085591123558202880;
            12'd2114: quad_corners = 76'd664085591123558202880;
            12'd2115: quad_corners = 76'd664085591123558202880;
            12'd2116: quad_corners = 76'd664085591123558202880;
            12'd2117: quad_corners = 76'd664085591123558202880;
            12'd2118: quad_corners = 76'd664085591123558202880;
            12'd2119: quad_corners = 76'd664085591123558202880;
            12'd2120: quad_corners = 76'd664085591123558202880;
            12'd2121: quad_corners = 76'd664085591123558202880;
            12'd2122: quad_corners = 76'd664085591123558202880;
            12'd2123: quad_corners = 76'd664085591123558202880;
            12'd2124: quad_corners = 76'd664085591123558202880;
            12'd2125: quad_corners = 76'd664085591123558202880;
            12'd2126: quad_corners = 76'd664085591123558202880;
            12'd2127: quad_corners = 76'd664085591123558202880;
            12'd2128: quad_corners = 76'd664085591123558202880;
            12'd2129: quad_corners = 76'd664085591123558202880;
            12'd2130: quad_corners = 76'd664085591123558202880;
            12'd2131: quad_corners = 76'd664085591123558202880;
            12'd2132: quad_corners = 76'd664085591123558202880;
            12'd2133: quad_corners = 76'd664085591123558202880;
            12'd2134: quad_corners = 76'd664085591123558202880;
            12'd2135: quad_corners = 76'd664085591123558202880;
            12'd2136: quad_corners = 76'd664085591123558202880;
            12'd2137: quad_corners = 76'd664085450386606718464;
            12'd2138: quad_corners = 76'd664085168912166878720;
            12'd2139: quad_corners = 76'd664084887437727038465;
            12'd2140: quad_corners = 76'd664084605963287198210;
            12'd2141: quad_corners = 76'd664084465226335713284;
            12'd2142: quad_corners = 76'd738015275234541500421;
            12'd2143: quad_corners = 76'd738303364873741727239;
            12'd2144: quad_corners = 76'd812378290070023369737;
            12'd2145: quad_corners = 76'd886453356004061803019;
            12'd2146: quad_corners = 76'd886741445642993594381;
            12'd2147: quad_corners = 76'd960960626764839447568;
            12'd2148: quad_corners = 76'd1035035692698609445395;
            12'd2149: quad_corners = 76'd1109254874095333206037;
            12'd2195: quad_corners = 76'd8854460964201444797 47;
            12'd2241: quad_corners = 76'd118059357801156896307 9;
            12’d2287: quad_corners = 76’d2737324012501838127170;
            12’d2288: quad_corners = 76’d2885474426118695869510;
            12’d2289: quad_corners = 76’d3033768954648751561291;
            12’d2290: quad_corners = 76’d3182063483453685160016;
            12’d2291: quad_corners = 76’d3330502127446694614613;
            12’d2292: quad_corners = 76’d3552583773283954774619;
            12’d2293: quad_corners = 76’d3552583773283954774619;
            12’d2294: quad_corners = 76’d3552583773283954774619;
            12’d2295: quad_corners = 76’d3552583773283954774619;
            12’d2296: quad_corners = 76’d3552583773283954774619;
            12’d2297: quad_corners = 76’d3552583773283954774619;
            12’d2298: quad_corners = 76’d3552583773283954774619;
            12’d2299: quad_corners = 76’d3552583773283954774619;
            12’d2300: quad_corners = 76’d3552583773283954774619;
            12’d2301: quad_corners = 76’d3552583773283954774619;
            12’d2302: quad_corners = 76’d3552583773283954774619;
            12’d2303: quad_corners = 76’d3552583773283954774619;
            12’d2304: quad_corners = 76’d1475741059602993446412;
            12’d2305: quad_corners = 76’d1475741059602993446412;
            12’d2306: quad_corners = 76’d1475741059602993446412;
            12’d2307: quad_corners = 76’d1475741059602993446412;
            12’d2308: quad_corners = 76’d1475741059602993446412;
            12’d2309: quad_corners = 76’d1475741059602993446412;
            12’d2310: quad_corners = 76’d1475741059602993446412;
            12’d2311: quad_corners = 76’d1475741059602993446412;
            12’d2312: quad_corners = 76’d1475741059602993446412;
            12’d2313: quad_corners = 76’d1475741059602993446412;
            12’d2314: quad_corners = 76’d1475741059602993446412;
            12’d2315: quad_corners = 76’d1475741059602993446412;
            12’d2316: quad_corners = 76’d1475741059602993446412;
            12’d2317: quad_corners = 76’d1475741059602993446412;
            12’d2318: quad_corners = 76’d1475741059602993446412;
            12’d2319: quad_corners = 76’d1475741059602993446412;
            12’d2320: quad_corners = 76’d1475741059602993446412;
            12’d2321: quad_corners = 76’d1475741059602993446412;
            12’d2322: quad_corners = 76’d1475741059602993446412;
            12’d2323: quad_corners = 76’d1475741059602993446412;
            12’d2324: quad_corners = 76’d1475741059602993446412;
            12’d2325: quad_corners = 76’d1475741059602993446412;
            12’d2326: quad_corners = 76’d1475741059602993446412;
            12’d2327: quad_corners = 76’d1475741059602993446412;
            12’d2328: quad_corners = 76’d1475741059602993446412;
            12’d2329: quad_corners = 76’d1475740778403431513613;
            12’d2330: quad_corners = 76’d1475740637666480029197;
            12’d2331: quad_corners = 76’d1475740356192308624910;
            12’d2332: quad_corners = 76’d1475740215455357140495;
            12'd2333: quad_corners = 76'd14757400749932835630 25;
            12’d2379: quad_corners = 76’d17708886819316378496l8;
```

179

```
            12'd2425: quad_corners = 76'd3920510128266518782047;
            12'd2471: quad_corners = 76'd25841295629002523930 07;
            12’d2517: quad_corners = 76’d2434970902334277484062;
            12'd2563: quad_corners = 76'd28776926177152087485581;
```

```
                    12'd2609: quad_corners = 76'd4138412006486627382361;
```

```
            12'd2655: quad_corners = 76'd3246627075880644312625;
            12'd2656: quad_corners = 76'd3320414052450897296947;
            12'd2657: quad_corners = 76'd3320414052726312074804;
            12'd2658: quad_corners = 76'd3320414052726848945717;
            12'd2659: quad_corners = 76'd3394201029297101930039;
            12'd2660: quad_corners = 76'd3467988005867086478905;
            12'd2661: quad_corners = 76'd3468276236243775061563;
            12'd2662: quad_corners = 76'd3542351443190179757629;
            12'd2663: quad_corners = 76'd3616570765324391874111;
            12'd2664: quad_corners = 76'd3690645972270528134721;
            12'd2665: quad_corners = 76'd3691078318110170480196;
            12'd2666: quad_corners = 76'd3765297640244382596678;
            12'd2667: quad_corners = 76'd3839516962378594713161;
            12'd2668: quad_corners = 76'd3987667375995720891980;
            12'd2669: quad_corners = 76'd4061886698129933007951;
            12'd2670: quad_corners = 76'd4136250135452220979794;
            12'd2671: quad_corners = 76'd4210469457586433096277;
            12'd2672: quad_corners = 76'd4358620011940779194456;
            12'd2673: quad_corners = 76'd4432983590000555521628;
            12'd2674: quad_corners = 76'd4507347168060331848800;
            12'd2675: quad_corners = 76'd4655641978340242158179;
            12'd2676: quad_corners = 76'd4803792673706954518631;
            12'd2677: quad_corners = 76'd4803792673706954518631;
            12'd2678: quad_corners = 76'd4803792673706954518631;
            12'd2679: quad_corners = 76'd4803792673706954518631;
            12'd2680: quad_corners = 76'd4803792673706954518631;
            12'd2681: quad_corners = 76'd4803792673706954518631;
            12'd2682: quad_corners = 76'd4803792673706954518631;
            12'd2683: quad_corners = 76'd4803792673706954518631;
            12'd2684: quad_corners = 76'd4803792673706954518631;
            12'd2685: quad_corners = 76'd4803792673706954518631;
            12'd2686: quad_corners = 76'd4803792673706954518631;
            12'd2687: quad_corners = 76'd4803792673706954518631;
            12'd2688: quad_corners = 76'd3689349071357599350324;
            12'd2689: quad_corners = 76'd3689349071357599350324;
            12'd2690: quad_corners = 76'd3689349071357599350324;
            12'd2691: quad_corners = 76'd3689349071357599350324;
            12'd2692: quad_corners = 76'd3689349071357599350324;
            12'd2693: quad_corners = 76'd3689349071357599350324;
            12'd2694: quad_corners = 76'd3689349071357599350324;
            12'd2695: quad_corners = 76'd3689349071357599350324;
            12'd2696: quad_corners = 76'd3689349071357599350324;
            12'd2697: quad_corners = 76'd3689349071357599350324;
            12'd2698: quad_corners = 76'd3689349071357599350324;
            12'd2699: quad_corners = 76'd3689349071357599350324;
            12'd2700: quad_corners = 76'd3689349071357599350324;
            12'd2701: quad_corners = 76'd36893490713575993503 24;
            12'd2747: quad_corners = 76'd50245772806511665444889;
            12'd2748: quad_corners = 76'd50245772806511665444889;
            12'd2749: quad_corners = 76'd50245772806511665444889;
            12'd2750: quad_corners = 76'd50245772806511665444889;
            12'd2751: quad_corners = 76'd50245772806511665444889;
            12'd2752: quad_corners = 76'd41320709272011410633228;
            12'd2753: quad_corners = 76'd41320709272011410633228;
            12'd2754: quad_corners = 76'd41320709272011410633228;
            12'd2755: quad_corners = 76'd41320709272011410633228;
            12'd2756: quad_corners = 76'd41320709272011410633228;
            12'd2757: quad_corners = 76'd41320709272011410633228;
            12'd2758: quad_corners = 76'd41320709272011410633228;
            12'd2759: quad_corners = 76'd41320709272011410633228;
            12'd2760: quad_corners = 76'd41320709272011410633228;
            12'd2761: quad_corners = 76'd41320709272011410633228;
            12'd2762: quad_corners = 76'd41320709272011410633228;
            12'd2763: quad_corners = 76'd41320709272011410633228;
            12'd2764: quad_corners = 76'd41320709272011410633228;
            12'd2765: quad_corners = 76'd41320709272011410633228;
            12'd2766: quad_corners = 76'd41320709272011410633228;
            12'd2767: quad_corners = 76'd41320709272011410633228;
            12'd2768: quad_corners = 76'd41320709272011410633228;
            12'd2769: quad_corners = 76'd41320709272011410633228;
            12'd2770: quad_corners = 76'd41320709272011410633228;
            12'd2771: quad_corners = 76'd41320709272011410633228;
            12'd2772: quad_corners = 76'd41320709272011410633228;
            12'd2773: quad_corners = 76'd41320709272011410633228;
            12'd2774: quad_corners = 76'd41320709272011410633228;
            12'd2775: quad_corners = 76'd41320709272011410633228;
            12'd2776: quad_corners = 76'd41320709272011410633228;
            12'd2777: quad_corners = 76'd41320707864644580142688;
            12'd2778: quad_corners = 76'd41320707867398727921255;
            12'd2779: quad_corners = 76'd41320707867404096630377;
            12'd2780: quad_corners = 76'd41320707870160928763500;
            12'd2781: quad_corners = 76'd41320707872915076542066;
            12'd2782: quad_corners = 76'd41320707872920445251199;
            12'd2783: quad_corners = 76'd41320707875674593029766;
            12'd2784: quad_corners = 76'd41320707878428740808333;
            12'd2785: quad_corners = 76'd41320707878434109517466;
            12'd2786: quad_corners = 76'd42058577644136639360677;
            12'd2787: quad_corners = 76'd42058577646890787139255;
            12'd2788: quad_corners = 76'd42058577649644934917822;
            12'd2789: quad_corners = 76'd42797888564476759895755;
            12'd2790: quad_corners = 76'd42800770870992424791777;
            12'd2791: quad_corners = 76'd43541522940453787397877;
            12'd2792: quad_corners = 76'd44282275009915150003977;
```

```
        12'd2793: quad_corners = 76'd4428659846831157345871;
        12'd2794: quad_corners = 76'd4502735053777293606481;
        12'd2795: quad_corners = 76'd4576954375911505722963;
        12'd2796: quad_corners = 76'd4651029582857641983573;
        12'd2797: quad_corners = 76'd4651461928697015893592;
        12'd2798: quad_corners = 76'd4725681250831228009562;
        12'd2799: quad_corners = 76'd4800044828890735901277;
        12'd2800: quad_corners = 76'd4874264432499924728416;
        12'd2801: quad_corners = 76'd5022270871666463406178;
        12'd2802: quad_corners = 76'd5096634590463459653221;
        12'd2803: quad_corners = 76'd5170854194347526386792;
        12'd2804: quad_corners = 76'd5245217772407034278508;
        12'd2805: quad_corners = 76'd5245217772407034278508;
        12'd2806: quad_corners = 76'd5245217772407034278508;
        12'd2807: quad_corners = 76'd5245217772407034278508;
        12'd2808: quad_corners = 76'd5245217772407034278508;
        12'd2809: quad_corners = 76'd5245217772407034278508;
        12'd2810: quad_corners = 76'd5245217772407034278508;
        12'd2811: quad_corners = 76'd5245217772407034278508;
        12'd2812: quad_corners = 76'd5245217772407034278508;
        12'd2813: quad_corners = 76'd5245217772407034278508;
        12'd2814: quad_corners = 76'd5245217772407034278508;
        12'd2815: quad_corners = 76'd5245217772407034278508;
        12'd2816: quad_corners = 76'd4574792782769536433733;
        12'd2817: quad_corners = 76'd4574792782769536433733;
        12'd2818: quad_corners = 76'd4574792782769536433733;
        12'd2819: quad_corners = 76'd4574792782769536433733;
        12'd2820: quad_corners = 76'd4574792782769536433733;
        12'd2821: quad_corners = 76'd4574792782769536433733;
        12'd2822: quad_corners = 76'd4574792782769536433733;
        12'd2823: quad_corners = 76'd4574792782769536433733;
        12'd2824: quad_corners = 76'd4574792782769536433733;
        12'd2825: quad_corners = 76'd4574792782769536433733;
        12'd2826: quad_corners = 76'd4574792782769536433733;
        12'd2827: quad_corners = 76'd4574792782769536433733;
        12'd2828: quad_corners = 76'd4574792782769536433733;
        12'd2829: quad_corners = 76'd4574792782769536433733;
        12'd2830: quad_corners = 76'd4574792782769536433733;
        12'd2831: quad_corners = 76'd4574792782769536433733;
        12'd2832: quad_corners = 76'd4574792782769536433733;
        12'd2833: quad_corners = 76'd4574792782769536433733;
        12'd2834: quad_corners = 76'd4574792782769536433733;
        12'd2835: quad_corners = 76'd4574792782769536433733;
        12'd2836: quad_corners = 76'd4574792782769536433733;
        12'd2837: quad_corners = 76'd4574792782769536433733;
        12'd2838: quad_corners = 76'd4574792782769536433733;
```

```
            12’d2839: quad_corners = 76’d4574792782769536433733;
            12’d2840: quad_corners = 76’d4574792782769536433733;
            12’d2841: quad_corners = 76’d4574792642307731291717;
            12’d2842: quad_corners = 76’d4574792642308268162629;
            12’d2843: quad_corners = 76’d4574792642583951375942;
            12’d2844: quad_corners = 76’d4574792642584488246854;
            12’d2845: quad_corners = 76’d4574792642859903024711;
            12’d2846: quad_corners = 76’d4574792643135586238023;
            12’d2847: quad_corners = 76’d4574792643136123108936;
            12’d2848: quad_corners = 76’d4574792643411537886793;
            12’d2849: quad_corners = 76’d4574792643686684229194;
            12’d2850: quad_corners = 76’d4648579619982059306571;
            12’d2851: quad_corners = 76’d4648579620257474084428;
            12’d2852: quad_corners = 76’d4648579620532888862285;
            12’d2853: quad_corners = 76’d4722510712290949267022;
            12’d2854: quad_corners = 76’d4722798942942515756624;
            12’d2855: quad_corners = 76’d4723087173318935903825;
            12’d2856: quad_corners = 76’d4797162380265072164435;
            12’d2857: quad_corners = 76’d4797450610916370218581;
            12’d2858: quad_corners = 76’d4871525817862774914646;
            12’d2859: quad_corners = 76’d4945601024808911175256;
            12’d2860: quad_corners = 76’d4946033370648285085274;
            12’d2861: quad_corners = 76’d5020108577594152910428;
            12’d2862: quad_corners = 76’d5094328181203341737055;
            12’d2863: quad_corners = 76’d5168547644075042208353;
            12’d2864: quad_corners = 76’d5242767106946474244195;
            12’d2865: quad_corners = 76’d5316986710555663070822;
            12’d2866: quad_corners = 76’d5391206173701973013608;
            12’d2867: quad_corners = 76’d5465425777311161840235;
            12’d2868: quad_corners = 76’d5539789496108158087278;
            12’d2869: quad_corners = 76’d5539789496108158087278;
            12’d2870: quad_corners = 76’d5539789496108158087278;
            12’d2871: quad_corners = 76’d5539789496108158087278;
            12’d2872: quad_corners = 76’d5539789496108158087278;
            12’d2873: quad_corners = 76’d5539789496108158087278;
            12’d2874: quad_corners = 76’d5539789496108158087278;
            12’d2875: quad_corners = 76’d5539789496108158087278;
            12’d2876: quad_corners = 76’d5539789496108158087278;
            12’d2877: quad_corners = 76’d5539789496108158087278;
            12’d2878: quad_corners = 76’d5539789496108158087278;
            12’d2879: quad_corners = 76’d5539789496108158087278;
            12’d2880: quad_corners = 76’d5091301614633037921870;
            12’d2881: quad_corners = 76’d5091301614633037921870;
            12’d2882: quad_corners = 76’d5091301614633037921870;
            12’d2883: quad_corners = 76’d5091301614633037921870;
            12’d2884: quad_corners = 76’d5091301614633037921870;
            12'd2885: quad_corners = 76'd5091301614633037921870;
```

```
                12'd2931: quad_corners = 76'd568621052444230110065 4;
```

```
            12'd2977: quad_corners = 76'd55340233308434352 69722;
            12'd3023: quad_corners = 76'd61981063948420716288898;
            12'd3024: quad_corners = 76'd61981063948420716288898;
            12'd3025: quad_corners = 76'd61981063948420716288898;
            12'd3026: quad_corners = 76'd61981063948420716288898;
            12'd3027: quad_corners = 76'd61981063948420716288898;
            12'd3028: quad_corners = 76'd61981063948420716288898;
            12'd3029: quad_corners = 76'd61981063948420716288898;
            12'd3030: quad_corners = 76'd61981063948420716288898;
            12'd3031: quad_corners = 76'd61981063948420716288898;
            12'd3032: quad_corners = 76'd61981063948420716288898;
            12'd3033: quad_corners = 76'd61243194188226487245778;
            12'd3034: quad_corners = 76'd61243192780859667247706;
            12'd3035: quad_corners = 76'd61243191376238936715222;
            12'd3036: quad_corners = 76'd61243191378993089736666;
            12'd3037: quad_corners = 76'd61243191378998463688666;
            12'd3038: quad_corners = 76'd61243191381752616710101;
            12'd3039: quad_corners = 76'd60505321621558387666690;
            12'd3040: quad_corners = 76'd60505321621563761618890;
            12'd3041: quad_corners = 76'd60506762776195983601633;
            12'd3042: quad_corners = 76'd60506762778950131380199;
            12'd3043: quad_corners = 76'd60508203930833574293488;
            12'd3044: quad_corners = 76'd60508203933587722072044;
            12'd3045: quad_corners = 76'd60509645088219944054777;
            12'd3046: quad_corners = 76'd60511086242852166037499;
            12'd3047: quad_corners = 76'd60512527394735608950788;
            12'd3048: quad_corners = 76'd60513969956742714481677;
            12'd3049: quad_corners = 76'd61253282281698202082322;
            12'd3050: quad_corners = 76'd61254723436330424059933;
            12'd3051: quad_corners = 76'd61256165998337529595944;
            12'd3052: quad_corners = 76'd61259049712222709330999;
            12'd3053: quad_corners = 76'd61260493681604698415166;
            12'd3054: quad_corners = 76'd62001247155689481150533;
            12'd3055: quad_corners = 76'd62002689717696586681422;
            12'd3056: quad_corners = 76'd62005574838956649969755;
            12'd3057: quad_corners = 76'd62008458552841829704811;
            12'd3058: quad_corners = 76'd62747772285169516499066;
            12'd3059: quad_corners = 76'd62750657406429579787399;
            12'd3060: quad_corners = 76'd63491410883263141587099;
            12'd3061: quad_corners = 76'd63491410883263141587099;
            12'd3062: quad_corners = 76'd63491410883263141587099;
            12'd3063: quad_corners = 76'd63491410883263141587099;
            12'd3064: quad_corners = 76'd63491410883263141587099;
            12'd3065: quad_corners = 76'd63491410883263141587099;
            12'd3066: quad_corners = 76'd63491410883263141587099;
            12'd3067: quad_corners = 76'd63491410883263141587099;
            12'd3068: quad_corners = 76'd63491410883263141587099;
            12'd3069: quad_corners = 76'd6349141088326314158709;
            12'd3070: quad_corners = 76'd6349141088326314158709;
            12'd3071: quad_corners = 76'd6349141088326314158709;
            12'd3072: quad_corners = 76'd6715336083845781190252;
            12'd3073: quad_corners = 76'd6715336083845781190252;
            12'd3074: quad_corners = 76'd6715336083845781190252;
            12'd3075: quad_corners = 76'd6715336083845781190252;
            12'd3076: quad_corners = 76'd6715336083845781190252;
            12'd3077: quad_corners = 76'd6715336083845781190252;
            12'd3078: quad_corners = 76'd6715336083845781190252;
            12'd3079: quad_corners = 76'd6715336083845781190252;
            12'd3080: quad_corners = 76'd6715336083845781190252;
            12'd3081: quad_corners = 76'd6715336083845781190252;
            12'd3082: quad_corners = 76'd6715336083845781190252;
            12'd3083: quad_corners = 76'd6715336083845781190252;
            12'd3084: quad_corners = 76'd6715336083845781190252;
            12'd3085: quad_corners = 76'd6715336083845781190252;
            12'd3086: quad_corners = 76'd6715336083845781190252;
            12'd3087: quad_corners = 76'd6715336083845781190252;
            12'd3088: quad_corners = 76'd6715336083845781190252;
            12'd3089: quad_corners = 76'd6715336083845781190252;
            12'd3090: quad_corners = 76'd6715336083845781190252;
            12'd3091: quad_corners = 76'd6715336083845781190252;
            12'd3092: quad_corners = 76'd6715336083845781190252;
            12'd3093: quad_corners = 76'd6715336083845781190252;
            12'd3094: quad_corners = 76'd6715336083845781190252;
            12'd3095: quad_corners = 76'd6715336083845781190252;
            12'd3096: quad_corners = 76'd6715336083845781190252;
            12'd3097: quad_corners = 76'd6715191828195632281708;
            12'd3098: quad_corners = 76'd6715191687458681321580;
            12'd3099: quad_corners = 76'd6715191687734096624236;
            12'd3100: quad_corners = 76'd6641260455514231601772;
            12'd3101: quad_corners = 76'd6641260455789646903916;
            12'd3102: quad_corners = 76'd6641260456065062730860;
            12'd3103: quad_corners = 76'd6641260315328111770732;
            12'd3104: quad_corners = 76'd6641260315603527072876;
            12'd3105: quad_corners = 76'd6567473339583835733100;
            12'd3106: quad_corners = 76'd6567617454772448984172;
            12'd3107: quad_corners = 76'd6567617595785084206188;
            12'd3108: quad_corners = 76'd6567617596060231072877;
            12'd3109: quad_corners = 76'd6567761711248575364205;
            12'd3110: quad_corners = 76'd6567761852261478496877;
            12'd3111: quad_corners = 76'd6567905967724432259694;
            12'd3112: quad_corners = 76'd6567906108737066956910;
            12'd3113: quad_corners = 76'd6568050364662899603567;
            12'd3114: quad_corners = 76'd6568194620863610156655;
            12'd3115: quad_corners = 76'd65681946211384880636OO;
            12'd3161: quad_corners = 76'd73062084932193016l8295;
            12'd3207: quad_corners = 76'd7971300364638276086403;
            12’d3253: quad_corners = 76’d7232135821475062798460;
            12’d3254: quad_corners = 76’d7232135821475062798460;
            12’d3255: quad_corners = 76’d7232135821475062798460;
            12’d3256: quad_corners = 76’d7232135821475062798460;
            12’d3257: quad_corners = 76’d7232135821475062798460;
            12’d3258: quad_corners = 76’d7232135821475062798460;
            12’d3259: quad_corners = 76’d7232135821475062798460;
            12’d3260: quad_corners = 76’d7232135821475062798460;
            12’d3261: quad_corners = 76’d7232135821475062798460;
            12’d3262: quad_corners = 76’d7232135821475062798460;
            12’d3263: quad_corners = 76’d7232135821475062798460;
            12’d3264: quad_corners = 76’d8562461285587240673934;
            12’d3265: quad_corners = 76’d8562461285587240673934;
            12’d3266: quad_corners = 76’d8562461285587240673934;
            12’d3267: quad_corners = 76’d8562461285587240673934;
            12’d3268: quad_corners = 76’d8562461285587240673934;
            12’d3269: quad_corners = 76’d8562461285587240673934;
            12’d3270: quad_corners = 76’d8562461285587240673934;
            12’d3271: quad_corners = 76’d8562461285587240673934;
            12’d3272: quad_corners = 76’d8562461285587240673934;
            12’d3273: quad_corners = 76’d8562461285587240673934;
            12’d3274: quad_corners = 76’d8562461285587240673934;
            12’d3275: quad_corners = 76’d8562461285587240673934;
            12’d3276: quad_corners = 76’d8562461285587240673934;
            12’d3277: quad_corners = 76’d8562461285587240673934;
            12’d3278: quad_corners = 76’d8562461285587240673934;
            12’d3279: quad_corners = 76’d8562461285587240673934;
            12’d3280: quad_corners = 76’d8562461285587240673934;
            12’d3281: quad_corners = 76’d8562461285587240673934;
            12’d3282: quad_corners = 76’d8562461285587240673934;
            12’d3283: quad_corners = 76’d8562461285587240673934;
            12’d3284: quad_corners = 76’d8562461285587240673934;
            12’d3285: quad_corners = 76’d8562461285587240673934;
            12’d3286: quad_corners = 76’d8562461285587240673934;
            12’d3287: quad_corners = 76’d8562461285587240673934;
            12’d3288: quad_corners = 76’d8562461285587240673934;
            12’d3289: quad_corners = 76’d8562317029662214382734;
            12’d3290: quad_corners = 76’d8562172914749553829517;
            12’d3291: quad_corners = 76’d8488241682804567238285;
            12’d3292: quad_corners = 76’d8488097567617028778124;
            12’d3293: quad_corners = 76’d8487953452704100313228;
            12’d3294: quad_corners = 76’d8414022361221723646603;
            12’d3295: quad_corners = 76’d8414022361496871037579;
            12’d3296: quad_corners = 76’d8340091270014225935498;
            12’d3297: quad_corners = 76’d8339947155101297470602;
            12’d3298: quad_corners = 76’d8266016063618652368009;
```

```
12’d3299: quad_corners = 76’d8265871948705723903113;
12’d3300: quad_corners = 76’d8191940997960567155848;
12’d3301: quad_corners = 76’d8191940998235714546824;
12’d3302: quad_corners = 76’d8118010047765167270535;
12’d3303: quad_corners = 76’d8117865932577360374406;
12’d3304: quad_corners = 76’d8043934982106813622918;
12’d3305: quad_corners = 76’d8043935122844302502021;
12’d3306: quad_corners = 76’d7970004031636267394693;
12’d3307: quad_corners = 76’d7969860057185680418436;
12’d3308: quad_corners = 76’d7896073221903209522308;
12’d3309: quad_corners = 76’d7822142271157784339075;
12’d3310: quad_corners = 76’d7822142552907638956162;
12’d3311: quad_corners = 76’d7748211602161944813186;
12’d3312: quad_corners = 76’d7748067627986235219073;
12’d3313: quad_corners = 76’d7674280933166105287297;
12’d3314: quad_corners = 76’d7600349982695289050752;
12’d3315: quad_corners = 76’d7526563287875159118975;
12’d3316: quad_corners = 76’d7526419313699181088895;
12’d3317: quad_corners = 76’d7526419313699181088895;
12’d3318: quad_corners = 76’d7526419313699181088895;
12’d3319: quad_corners = 76’d7526419313699181088895;
12’d3320: quad_corners = 76’d7526419313699181088895;
12’d3321: quad_corners = 76’d7526419313699181088895;
12’d3322: quad_corners = 76’d7526419313699181088895;
12’d3323: quad_corners = 76’d7526419313699181088895;
12’d3324: quad_corners = 76’d7526419313699181088895;
12’d3325: quad_corners = 76’d7526419313699181088895;
12’d3326: quad_corners = 76’d7526419313699181088895;
12’d3327: quad_corners = 76’d7526419313699181088895;
12’d3328: quad_corners = 76’d9227553297744240892571;
12’d3329: quad_corners = 76’d9227553297744240892571;
12’d3330: quad_corners = 76’d9227553297744240892571;
12’d3331: quad_corners = 76’d9227553297744240892571;
12’d3332: quad_corners = 76’d9227553297744240892571;
12’d3333: quad_corners = 76’d9227553297744240892571;
12’d3334: quad_corners = 76’d9227553297744240892571;
12’d3335: quad_corners = 76’d9227553297744240892571;
12’d3336: quad_corners = 76’d9227553297744240892571;
12’d3337: quad_corners = 76’d9227553297744240892571;
12’d3338: quad_corners = 76’d9227553297744240892571;
12’d3339: quad_corners = 76’d9227553297744240892571;
12’d3340: quad_corners = 76’d9227553297744240892571;
12’d3341: quad_corners = 76’d9227553297744240892571;
12’d3342: quad_corners = 76’d9227553297744240892571;
12’d3343: quad_corners = 76’d9227553297744240892571;
12’d3344: quad_corners = 76’d9227553297744240892571;
```

```
            12'd3345: quad_corners = 76'd9227553297744240892571;
            12'd3346: quad_corners = 76'd9227553297744240892571;
            12'd3347: quad_corners = 76'd9227553297744240892571;
            12'd3348: quad_corners = 76'd9227553297744240892571;
            12'd3349: quad_corners = 76'd9227553297744240892571;
            12'd3350: quad_corners = 76'd9227553297744240892571;
            12'd3351: quad_corners = 76'd9227553297744240892571;
            12'd3352: quad_corners = 76'd9227553297744240892571;
            12'd3353: quad_corners = 76'd9227409042094092508314;
            12'd3354: quad_corners = 76'd9227264926906554048154;
            12'd3355: quad_corners = 76'd9153333694961567457433;
            12'd3356: quad_corners = 76'd9153045464585685229720;
            12'd3357: quad_corners = 76'd9079114373378186470040;
            12'd3358: quad_corners = 76'd9078970258190380098199;
            12'd3359: quad_corners = 76'd9005039166982613427350;
            12'd3360: quad_corners = 76'd9004750936606730675349;
            12'd3361: quad_corners = 76'd8930819845398964003988;
            12'd3362: quad_corners = 76'd8930675730211157107860;
            12'd3363: quad_corners = 76'd8856744779740878791827;
            12'd3364: quad_corners = 76'd8782669573069889922194;
            12'd3365: quad_corners = 76'd8782525458156960933009;
            12'd3366: quad_corners = 76'd8708594507411536274576;
            12'd3367: quad_corners = 76'd8634663416203500643471;
            12'd3368: quad_corners = 76'd8634519441752914190990;
            12'd3369: quad_corners = 76'd8560444375819413152397;
            12'd3370: quad_corners = 76'd8486513425348866400396;
            12'd3371: quad_corners = 76'd8412582333865952861835;
            12'd3372: quad_corners = 76'd8412438359690244316298;
            12'd3373: quad_corners = 76'd8338507408944551221897;
            12'd3374: quad_corners = 76'd8264432483748538538120;
            12'd3375: quad_corners = 76'd8190501533277723350663;
            12'd3376: quad_corners = 76'd8116570582532029731974;
            12'd3377: quad_corners = 76'd8042783746974411444869;
            12'd3378: quad_corners = 76'd7968997052429159419524;
            12'd3379: quad_corners = 76'd7895210216871541132419;
            12'd3380: quad_corners = 76'd7821423522051142764673;
            12'd3381: quad_corners = 76'd7821423522051142764673;
            12'd3382: quad_corners = 76'd7821423522051142764673;
            12'd3383: quad_corners = 76'd7821423522051142764673;
            12'd3384: quad_corners = 76'd7821423522051142764673;
            12'd3385: quad_corners = 76'd7821423522051142764673;
            12'd3386: quad_corners = 76'd7821423522051142764673;
            12'd3387: quad_corners = 76'd7821423522051142764673;
            12'd3388: quad_corners = 76'd7821423522051142764673;
            12'd3389: quad_corners = 76'd7821423522051142764673;
            12'd3390: quad_corners = 76'd7821423522051142764673;
            12'd3391: quad_corners = 76'd7821423522051142764673;
            12'd3437: quad_corners = 76'd8338507408944551221897;
            12'd3438: quad_corners = 76'd8264432483748538538120;
            12'd3439: quad_corners = 76'd8190501533277723350663;
            12'd3440: quad_corners = 76'd8116570582532029731974;
            12'd3441: quad_corners = 76'd8042783746974411444869;
            12'd3442: quad_corners = 76'd7968997052429159419524;
            12'd3443: quad_corners = 76'd7895210216871541132419;
            12'd3444: quad_corners = 76'd7821423522051142764673;
            12'd3445: quad_corners = 76'd7821423522051142764673;
            12'd3446: quad_corners = 76'd7821423522051142764673;
            12'd3447: quad_corners = 76'd7821423522051142764673;
            12'd3448: quad_corners = 76'd7821423522051142764673;
            12'd3449: quad_corners = 76'd7821423522051142764673;
            12'd3450: quad_corners = 76'd7821423522051142764673;
            12'd3451: quad_corners = 76'd7821423522051142764673;
            12'd3452: quad_corners = 76'd7821423522051142764673;
            12'd3453: quad_corners = 76'd7821423522051142764673;
            12'd3454: quad_corners = 76'd7821423522051142764673;
            12'd3455: quad_corners = 76'd7821423522051142764673;
            12'd3456: quad_corners = 76'd9227553297744240892571;
            12'd3457: quad_corners = 76'd9227553297744240892571;
            12'd3458: quad_corners = 76'd9227553297744240892571;
            12'd3459: quad_corners = 76'd9227553297744240892571;
            12'd3460: quad_corners = 76'd9227553297744240892571;
            12'd3461: quad_corners = 76'd9227553297744240892571;
            12'd3462: quad_corners = 76'd9227553297744240892571;
            12'd3463: quad_corners = 76'd9227553297744240892571;
            12'd3464: quad_corners = 76'd9227553297744240892571;
            12'd3465: quad_corners = 76'd9227553297744240892571;
            12'd3466: quad_corners = 76'd9227553297744240892571;
            12'd3467: quad_corners = 76'd9227553297744240892571;
            12'd3468: quad_corners = 76'd9227553297744240892571;
            12'd3469: quad_corners = 76'd9227553297744240892571;
            12'd3470: quad_corners = 76'd9227553297744240892571;
            12'd3471: quad_corners = 76'd9227553297744240892571;
            12'd3472: quad_corners = 76'd9227553297744240892571;
            12'd3473: quad_corners = 76'd9227553297744240892571;
            12'd3474: quad_corners = 76'd9227553297744240892571;
            12'd3475: quad_corners = 76'd9227553297744240892571;
            12'd3476: quad_corners = 76'd9227553297744240892571;
            12'd3477: quad_corners = 76'd9227553297744240892571;
            12'd3478: quad_corners = 76'd9227553297744240892571;
            12'd3479: quad_corners = 76'd9227553297744240892571;
            12'd3480: quad_corners = 76'd9227553297744240892571;
            12'd3481: quad_corners = 76'd9227409042094092508314;
            12'd3482: quad_corners = 76'd9227264926906554048154;
            12'd3483: quad_corners = 76'd915333369496156745743 3;
```

```
                12'd3529: quad_corners = 76'd922755329774424089257l;
                12’d3575: quad_corners = 76’d7821423522051142764673;
                12’d3576: quad_corners = 76’d7821423522051142764673;
                12’d3577: quad_corners = 76’d7821423522051142764673;
                12’d3578: quad_corners = 76’d7821423522051142764673;
                12’d3579: quad_corners = 76’d7821423522051142764673;
                12’d3580: quad_corners = 76’d7821423522051142764673;
                12’d3581: quad_corners = 76’d7821423522051142764673;
                12’d3582: quad_corners = 76’d7821423522051142764673;
                12’d3583: quad_corners = 76’d7821423522051142764673;
                12’d3584: quad_corners = 76’d9227553297744240892571;
                12’d3585: quad_corners = 76’d9227553297744240892571;
                12’d3586: quad_corners = 76’d9227553297744240892571;
                12’d3587: quad_corners = 76’d9227553297744240892571;
                12’d3588: quad_corners = 76’d9227553297744240892571;
                12’d3589: quad_corners = 76’d9227553297744240892571;
                12’d3590: quad_corners = 76’d9227553297744240892571;
                12’d3591: quad_corners = 76’d9227553297744240892571;
                12’d3592: quad_corners = 76’d9227553297744240892571;
                12’d3593: quad_corners = 76’d9227553297744240892571;
                12’d3594: quad_corners = 76’d9227553297744240892571;
                12’d3595: quad_corners = 76’d9227553297744240892571;
                12’d3596: quad_corners = 76’d9227553297744240892571;
                12’d3597: quad_corners = 76’d9227553297744240892571;
                12’d3598: quad_corners = 76’d9227553297744240892571;
                12’d3599: quad_corners = 76’d9227553297744240892571;
                12’d3600: quad_corners = 76’d9227553297744240892571;
                12’d3601: quad_corners = 76’d9227553297744240892571;
                12’d3602: quad_corners = 76’d9227553297744240892571;
                12’d3603: quad_corners = 76’d9227553297744240892571;
                12’d3604: quad_corners = 76’d9227553297744240892571;
                12’d3605: quad_corners = 76’d9227553297744240892571;
                12’d3606: quad_corners = 76’d9227553297744240892571;
                12’d3607: quad_corners = 76’d9227553297744240892571;
                12’d3608: quad_corners = 76’d9227553297744240892571;
                12’d3609: quad_corners = 76’d9227409042094092508314;
                12’d3610: quad_corners = 76’d9227264926906554048154;
                12’d3611: quad_corners = 76’d9153333694961567457433;
                12’d3612: quad_corners = 76’d9153045464585685229720;
                12’d3613: quad_corners = 76’d9079114373378186470040;
                12’d3614: quad_corners = 76’d9078970258190380098199;
                12’d3615: quad_corners = 76’d9005039166982613427350;
                12’d3616: quad_corners = 76’d9004750936606730675349;
                12’d3617: quad_corners = 76’d8930819845398964003988;
                12’d3618: quad_corners = 76’d8930675730211157107860;
                12’d3619: quad_corners = 76’d8856744779740878791827;
                12’d3620: quad_corners = 76’d8782669573069889922194;
```

```
            12'd3621: quad_corners = 76'd8782525458156960933009;
            12'd3622: quad_corners = 76'd8708594507411536274576;
            12'd3623: quad_corners = 76'd8634663416203500643471;
            12'd3624: quad_corners = 76'd8634519441752914190990;
            12'd3625: quad_corners = 76'd8560444375819413152397;
            12'd3626: quad_corners = 76'd8486513425348866400396;
            12'd3627: quad_corners = 76'd8412582333865952861835;
            12'd3628: quad_corners = 76'd8412438359690244316298;
            12'd3629: quad_corners = 76'd8338507408944551221897;
            12'd3630: quad_corners = 76'd8264432483748538538120;
            12'd3631: quad_corners = 76'd8190501533277723350663;
            12'd3632: quad_corners = 76'd8116570582532029731974;
            12'd3633: quad_corners = 76'd8042783746974411444869;
            12'd3634: quad_corners = 76'd7968997052429159419524;
            12'd3635: quad_corners = 76'd7895210216871541132419;
            12'd3636: quad_corners = 76'd7821423522051142764673;
            12'd3637: quad_corners = 76'd7821423522051142764673;
            12'd3638: quad_corners = 76'd7821423522051142764673;
            12'd3639: quad_corners = 76'd7821423522051142764673;
            12'd3640: quad_corners = 76'd7821423522051142764673;
            12'd3641: quad_corners = 76'd7821423522051142764673;
            12'd3642: quad_corners = 76'd7821423522051142764673;
            12'd3643: quad_corners = 76'd7821423522051142764673;
            12'd3644: quad_corners = 76'd7821423522051142764673;
            12'd3645: quad_corners = 76'd7821423522051142764673;
            12'd3646: quad_corners = 76'd7821423522051142764673;
            12'd3647: quad_corners = 76'd7821423522051142764673;
            12'd3648: quad_corners = 76'd9227553297744240892571;
            12'd3649: quad_corners = 76'd9227553297744240892571;
            12'd3650: quad_corners = 76'd9227553297744240892571;
            12'd3651: quad_corners = 76'd9227553297744240892571;
            12'd3652: quad_corners = 76'd9227553297744240892571;
            12'd3653: quad_corners = 76'd9227553297744240892571;
            12'd3654: quad_corners = 76'd9227553297744240892571;
            12'd3655: quad_corners = 76'd9227553297744240892571;
            12'd3656: quad_corners = 76'd9227553297744240892571;
            12'd3657: quad_corners = 76'd9227553297744240892571;
            12'd3658: quad_corners = 76'd9227553297744240892571;
            12'd3659: quad_corners = 76'd9227553297744240892571;
            12'd3660: quad_corners = 76'd9227553297744240892571;
            12'd3661: quad_corners = 76'd9227553297744240892571;
            12'd3662: quad_corners = 76'd9227553297744240892571;
            12'd3663: quad_corners = 76'd9227553297744240892571;
            12'd3664: quad_corners = 76'd9227553297744240892571;
            12'd3665: quad_corners = 76'd9227553297744240892571;
            12'd3666: quad_corners = 76'd9227553297744240892571;
            12'd3667: quad_corners = 76'd922755329774424089257 1;
            12'd3713: quad_corners = 76'd9227553297744240892571;
```

```
                12’d3759: quad_corners = 76’d819050153327772335063;
                12’d3760: quad_corners = 76’d811657058253202973197 4;
```

```
            12’d3805: quad_corners = 76’d907911437337818647004 0;
```

210

```
        12’d3851: quad_corners = 76’d922755329774424089257l;
```

```
            12'd3897: quad_corners = 76'd7821423522051142764673;
            12'd3898: quad_corners = 76'd7821423522051142764673;
            12'd3899: quad_corners = 76'd7821423522051142764673;
            12'd3900: quad_corners = 76'd7821423522051142764673;
            12'd3901: quad_corners = 76'd7821423522051142764673;
            12'd3902: quad_corners = 76'd7821423522051142764673;
            12'd3903: quad_corners = 76'd7821423522051142764673;
            12'd3904: quad_corners = 76'd9227553297744240892571;
            12'd3905: quad_corners = 76'd9227553297744240892571;
            12'd3906: quad_corners = 76'd9227553297744240892571;
            12'd3907: quad_corners = 76'd9227553297744240892571;
            12'd3908: quad_corners = 76'd9227553297744240892571;
            12'd3909: quad_corners = 76'd9227553297744240892571;
            12'd3910: quad_corners = 76'd9227553297744240892571;
            12'd3911: quad_corners = 76'd9227553297744240892571;
            12'd3912: quad_corners = 76'd9227553297744240892571;
            12'd3913: quad_corners = 76'd9227553297744240892571;
            12'd3914: quad_corners = 76'd9227553297744240892571;
            12'd3915: quad_corners = 76'd9227553297744240892571;
            12'd3916: quad_corners = 76'd9227553297744240892571;
            12'd3917: quad_corners = 76'd9227553297744240892571;
            12'd3918: quad_corners = 76'd9227553297744240892571;
            12'd3919: quad_corners = 76'd9227553297744240892571;
            12'd3920: quad_corners = 76'd9227553297744240892571;
            12'd3921: quad_corners = 76'd9227553297744240892571;
            12'd3922: quad_corners = 76'd9227553297744240892571;
            12'd3923: quad_corners = 76'd9227553297744240892571;
            12'd3924: quad_corners = 76'd9227553297744240892571;
            12'd3925: quad_corners = 76'd9227553297744240892571;
            12'd3926: quad_corners = 76'd9227553297744240892571;
            12'd3927: quad_corners = 76'd9227553297744240892571;
            12'd3928: quad_corners = 76'd9227553297744240892571;
            12'd3929: quad_corners = 76'd9227409042094092508314;
            12'd3930: quad_corners = 76'd9227264926906554048154;
            12'd3931: quad_corners = 76'd9153333694961567457433;
            12'd3932: quad_corners = 76'd9153045464585685229720;
            12'd3933: quad_corners = 76'd9079114373378186470040;
            12'd3934: quad_corners = 76'd9078970258190380098199;
            12'd3935: quad_corners = 76'd9005039166982613427350;
            12'd3936: quad_corners = 76'd9004750936606730675349;
            12'd3937: quad_corners = 76'd8930819845398964003988;
            12'd3938: quad_corners = 76'd8930675730211157107860;
            12'd3939: quad_corners = 76'd8856744779740878791827;
            12'd3940: quad_corners = 76'd8782669573069889922194;
            12'd3941: quad_corners = 76'd8782525458156960933009;
            12'd3942: quad_corners = 76'd8708594507411536274576;
            12’d3943: quad_corners = 76’d8634663416203500643471;
            12’d3944: quad_corners = 76’d8634519441752914190990;
            12’d3945: quad_corners = 76’d8560444375819413152397;
            12’d3946: quad_corners = 76’d8486513425348866400396;
            12’d3947: quad_corners = 76’d8412582333865952861835;
            12’d3948: quad_corners = 76’d8412438359690244316298;
            12’d3949: quad_corners = 76’d8338507408944551221897;
            12’d3950: quad_corners = 76’d8264432483748538538120;
            12’d3951: quad_corners = 76’d8190501533277723350663;
            12’d3952: quad_corners = 76’d8116570582532029731974;
            12’d3953: quad_corners = 76’d8042783746974411444869;
            12’d3954: quad_corners = 76’d7968997052429159419524;
            12’d3955: quad_corners = 76’d7895210216871541132419;
            12’d3956: quad_corners = 76’d7821423522051142764673;
            12’d3957: quad_corners = 76’d7821423522051142764673;
            12’d3958: quad_corners = 76’d7821423522051142764673;
            12’d3959: quad_corners = 76’d7821423522051142764673;
            12’d3960: quad_corners = 76’d7821423522051142764673;
            12’d3961: quad_corners = 76’d7821423522051142764673;
            12’d3962: quad_corners = 76’d7821423522051142764673;
            12’d3963: quad_corners = 76’d7821423522051142764673;
            12’d3964: quad_corners = 76’d7821423522051142764673;
            12’d3965: quad_corners = 76’d7821423522051142764673;
            12’d3966: quad_corners = 76’d7821423522051142764673;
            12’d3967: quad_corners = 76’d7821423522051142764673;
            12’d3968: quad_corners = 76’d9227553297744240892571;
            12’d3969: quad_corners = 76’d9227553297744240892571;
            12’d3970: quad_corners = 76’d9227553297744240892571;
            12’d3971: quad_corners = 76’d9227553297744240892571;
            12’d3972: quad_corners = 76’d9227553297744240892571;
            12’d3973: quad_corners = 76’d9227553297744240892571;
            12’d3974: quad_corners = 76’d9227553297744240892571;
            12’d3975: quad_corners = 76’d9227553297744240892571;
            12’d3976: quad_corners = 76’d9227553297744240892571;
            12’d3977: quad_corners = 76’d9227553297744240892571;
            12’d3978: quad_corners = 76’d9227553297744240892571;
            12’d3979: quad_corners = 76’d9227553297744240892571;
            12’d3980: quad_corners = 76’d9227553297744240892571;
            12’d3981: quad_corners = 76’d9227553297744240892571;
            12’d3982: quad_corners = 76’d9227553297744240892571;
            12’d3983: quad_corners = 76’d9227553297744240892571;
            12’d3984: quad_corners = 76’d9227553297744240892571;
            12’d3985: quad_corners = 76’d9227553297744240892571;
            12’d3986: quad_corners = 76’d9227553297744240892571;
            12’d3987: quad_corners = 76’d9227553297744240892571;
            12’d3988: quad_corners = 76’d9227553297744240892571;
            12’d3989: quad_corners = 76’d9227553297744240892571;
            12’d3990: quad_corners = 76’d9227553297744240892571;
            12’d3991: quad_corners = 76’d9227553297744240892571;
            12’d3992: quad_corners = 76’d9227553297744240892571;
            12’d3993: quad_corners = 76’d9227409042094092508314;
            12’d3994: quad_corners = 76’d9227264926906554048154;
            12’d3995: quad_corners = 76’d9153333694961567457433;
            12’d3996: quad_corners = 76’d9153045464585685229720;
            12’d3997: quad_corners = 76’d9079114373378186470040;
            12’d3998: quad_corners = 76’d9078970258190380098199;
            12’d3999: quad_corners = 76’d9005039166982613427350;
            12’d4000: quad_corners = 76’d9004750936606730675349;
            12’d4001: quad_corners = 76’d8930819845398964003988;
            12’d4002: quad_corners = 76’d8930675730211157107860;
            12’d4003: quad_corners = 76’d8856744779740878791827;
            12’d4004: quad_corners = 76’d8782669573069889922194;
            12’d4005: quad_corners = 76’d8782525458156960933009;
            12’d4006: quad_corners = 76’d8708594507411536274576;
            12’d4007: quad_corners = 76’d8634663416203500643471;
            12’d4008: quad_corners = 76’d8634519441752914190990;
            12’d4009: quad_corners = 76’d8560444375819413152397;
            12’d4010: quad_corners = 76’d8486513425348866400396;
            12’d4011: quad_corners = 76’d8412582333865952861835;
            12’d4012: quad_corners = 76’d8412438359690244316298;
            12’d4013: quad_corners = 76’d8338507408944551221897;
            12’d4014: quad_corners = 76’d8264432483748538538120;
            12’d4015: quad_corners = 76’d8190501533277723350663;
            12’d4016: quad_corners = 76’d8116570582532029731974;
            12’d4017: quad_corners = 76’d8042783746974411444869;
            12’d4018: quad_corners = 76’d7968997052429159419524;
            12’d4019: quad_corners = 76’d7895210216871541132419;
            12’d4020: quad_corners = 76’d7821423522051142764673;
            12’d4021: quad_corners = 76’d7821423522051142764673;
            12’d4022: quad_corners = 76’d7821423522051142764673;
            12’d4023: quad_corners = 76’d7821423522051142764673;
            12’d4024: quad_corners = 76’d7821423522051142764673;
            12’d4025: quad_corners = 76’d7821423522051142764673;
            12’d4026: quad_corners = 76’d7821423522051142764673;
            12’d4027: quad_corners = 76’d7821423522051142764673;
            12’d4028: quad_corners = 76’d7821423522051142764673;
            12’d4029: quad_corners = 76’d7821423522051142764673;
            12’d4030: quad_corners = 76’d7821423522051142764673;
            12’d4031: quad_corners = 76’d7821423522051142764673;
            12’d4032: quad_corners = 76’d9227553297744240892571;
            12’d4033: quad_corners = 76’d9227553297744240892571;
            12’d4034: quad_corners = 76’d9227553297744240892571;
```

```
                        12'd4035: quad_corners = 76'd9227553297744240892571;
                        12'd4036: quad_corners = 76'd9227553297744240892571;
                        12'd4037: quad_corners = 76'd9227553297744240892571;
                        12'd4038: quad_corners = 76'd9227553297744240892571;
                        12'd4039: quad_corners = 76'd9227553297744240892571;
                        12'd4040: quad_corners = 76'd9227553297744240892571;
                        12'd4041: quad_corners = 76'd9227553297744240892571;
                        12'd4042: quad_corners = 76'd9227553297744240892571;
                        12'd4043: quad_corners = 76'd9227553297744240892571;
                        12'd4044: quad_corners = 76'd9227553297744240892571;
                        12'd4045: quad_corners = 76'd9227553297744240892571;
                        12'd4046: quad_corners = 76'd9227553297744240892571;
                        12'd4047: quad_corners = 76'd9227553297744240892571;
                        12'd4048: quad_corners = 76'd9227553297744240892571;
                        12'd4049: quad_corners = 76'd9227553297744240892571;
                        12'd4050: quad_corners = 76'd9227553297744240892571;
                        12'd4051: quad_corners = 76'd9227553297744240892571;
                        12'd4052: quad_corners = 76'd9227553297744240892571;
                        12'd4053: quad_corners = 76'd9227553297744240892571;
                        12'd4054: quad_corners = 76'd9227553297744240892571;
                        12'd4055: quad_corners = 76'd9227553297744240892571;
                        12'd4056: quad_corners = 76'd9227553297744240892571;
                        12'd4057: quad_corners = 76'd9227409042094092508314;
                        12'd4058: quad_corners = 76'd9227264926906554048154;
                        12'd4059: quad_corners = 76'd9153333694961567457433;
                        12'd4060: quad_corners = 76'd9153045464585685229720;
                        12'd4061: quad_corners = 76'd9079114373378186470040;
                        12'd4062: quad_corners = 76'd9078970258190380098199;
                        12'd4063: quad_corners = 76'd9005039166982613427350;
                        12'd4064: quad_corners = 76'd9004750936606730675349;
                        12'd4065: quad_corners = 76'd8930819845398964003988;
                        12'd4066: quad_corners = 76'd8930675730211157107860;
                        12'd4067: quad_corners = 76'd8856744779740878791827;
                        12'd4068: quad_corners = 76'd8782669573069889922194;
                        12'd4069: quad_corners = 76'd8782525458156960933009;
                        12'd4070: quad_corners = 76'd8708594507411536274576;
                        12'd4071: quad_corners = 76'd8634663416203500643471;
                        12'd4072: quad_corners = 76'd8634519441752914190990;
                        12'd4073: quad_corners = 76'd8560444375819413152397;
                        12'd4074: quad_corners = 76'd8486513425348866400396;
                        12'd4075: quad_corners = 76'd8412582333865952861835;
                        12'd4076: quad_corners = 76'd8412438359690244316298;
                        12'd4077: quad_corners = 76'd8338507408944551221897;
                        12'd4078: quad_corners = 76'd8264432483748538538120;
                        12'd4079: quad_corners = 76'd8190501533277723350663;
                        12'd4080: quad_corners = 76'd8116570582532029731974;
```

```
                12'd4081: quad_corners = 76'd8042783746974411444869;
```

A.3.3 accel_lut.jl

```
#=
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

This script generates an accel_lut.v file.
accel_lut.v contains a verilog implementation of a lookup table,
which takes in an accelerometer reading (6 bit x dir, 6 bit y dir),
and looks up a 76 bit value (4 corners of quadrilateral)
This script requires a input file accel_lut.txt containing data points.
It then interpolates the data points using 2D splines,
and creates the desired lookup table.

Format of accel_lut.txt:
```

```
x_accel, y_accel, x1, y1, x2, y2, x3, y3, x4, y4

i.e it is a csv file, each line denoting a reading
for ease of entry, all values are hex (just read off hex display)
leading zeros don’t have to be specified, but can be if desired

NOTE: accel_lut.v and accel_lut.txt are suggested names
General installation:
1) Install julia
2) open Julia interpreter, i.e julia at cmd line
3) install gfortran (gcc frontend for fortran), needed to compile interpolation package
4) install Dierckx (for the spline interpolation) by ‘Pkg.add("Dierckx")’ at julia prompt
5) Dierckx details (docs, installation help, etc): https://github.com/kbarbary/Dierckx.jl

General usage:
accel_lut(input_path, output_path) at julia prompt

input_path is path to the csv file, and output_path is path to the desired .v file
NOTE: in order to run the command, you first need to include this file, so type:
include("accel_lut.jl") at the julia prompt prior to running the above
NOTE: THIS CODE WILL OVERWRITE THE FILE AT OUTPUT_PATH!!!

Suggested usage:
accel_lut("./accel_lut.txt", "./accel_lut.v")
=#

using Dierckx # interpolation package

function read_file(path)
    return readcsv(path, String)
end

function parse_data(path)
    raw_data = read_file(path)
    num_samples = size(raw_data)[1]
    x_accel = zeros(Int64, num_samples)
    y_accel = zeros(Int64, num_samples)
    x1 = zeros(Int64, num_samples)
    y1 = zeros(Int64, num_samples)
    x2 = zeros(Int64, num_samples)
    y2 = zeros(Int64, num_samples)
    x3 = zeros(Int64, num_samples)
    y3 = zeros(Int64, num_samples)
    x4 = zeros(Int64, num_samples)
    y4 = zeros(Int64, num_samples)
    base = 16
    for i = 1:num_samples
        x_accel[i] = parseint(raw_data[i,1], base)
        y_accel[i] = parseint(raw_data[i,2], base)
        x1[i] = parseint(raw_data[i,3], base)
        y1[i] = parseint(raw_data[i,4], base)
        x2[i] = parseint(raw_data[i,5], base)
        y2[i] = parseint(raw_data[i,6], base)
        x3[i] = parseint(raw_data[i,7], base)
        y3[i] = parseint(raw_data[i,8], base)
        x4[i] = parseint(raw_data[i,9], base)
        y4[i] = parseint(raw_data[i,10], base)
    end
    return float(x_accel), float(y_accel), float(x1), float(y1), float(x2), float(y2), floa
end

function saturate!(vec, low, upp)
    for i=1:length(vec)
        if (vec[i] < low)
            vec[i] = low
        elseif (vec[i] > upp)
            vec[i] = upp
        end
    end
    return vec
end

function write_file(path, x_accel, y_accel, x1, y1, x2, y2, x3, y3, x4, y4)
    # compute grid
    x = zeros(2^12)
    y = zeros(2^12)
    quad_corners = zeros(Int128, 2^12)
    for i=0:2^12-1
        x[i+1] = i >> 6
        y[i+1] = i & ((1 << 6) - 1)
    end

    # do the spline interpolation
    # we do linear fits for now
    # as we add more points, we can do something more sophisticated
    x_deg = 2;
    y_deg = 2;
    # we also use a smoothing factor
    # this trades off exact interpolation vs weighted least squares
    # for more details, see doc at: https://github.com/kbarbary/Dierckx.jl
    smooth_factor = 454.0;
    spline_x1 = Spline2D(x_accel, y_accel, x1; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_x2 = Spline2D(x_accel, y_accel, x2; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_x3 = Spline2D(x_accel, y_accel, x3; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_x4 = Spline2D(x_accel, y_accel, x4; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_y1 = Spline2D(x_accel, y_accel, y1; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_y2 = Spline2D(x_accel, y_accel, y2; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_y3 = Spline2D(x_accel, y_accel, y3; kx=x_deg, ky=y_deg, s=smooth_factor)
    spline_y4 = Spline2D(x_accel, y_accel, y4; kx=x_deg, ky=y_deg, s=smooth_factor)
    x1_interp = int128(evaluate(spline_x1, x, y))
    x2_interp = int128(evaluate(spline_x2, x, y))
    x3_interp = int128(evaluate(spline_x3, x, y))
    x4_interp = int128(evaluate(spline_x4, x, y))
    y1_interp = int128(evaluate(spline_y1, x, y))
    y2_interp = int128(evaluate(spline_y2, x, y))
    y3_interp = int128(evaluate(spline_y3, x, y))
    y4_interp = int128(evaluate(spline_y4, x, y))

    # threshold x, y coords at appropriate values
    # this is to guarantee we are not putting garbage into the lut
    low_x = 0
    low_y = 0
    upp_x = 639
    upp_y = 479
    saturate!(x1_interp, low_x, upp_x)
    saturate!(x2_interp, low_x, upp_x)
    saturate!(x3_interp, low_x, upp_x)
    saturate!(x4_interp, low_x, upp_x)
    saturate!(y1_interp, low_y, upp_y)
    saturate!(y2_interp, low_y, upp_y)
    saturate!(y3_interp, low_y, upp_y)
    saturate!(y4_interp, low_y, upp_y)

    # compute quad_corners
    for i=1:2^12
        quad_corners[i] = y4_interp[i] + (x4_interp[i] << 9) + (y3_interp[i] << 19) + (x3_i
        quad_corners[i] += (y2_interp[i] << 38) + (x2_interp[i] << 47) + (y1_interp[i] << 57
    end

    # write header
    comment_head = "///////////////////////////////////////////////////////////////////////
    comment_body1 = "//This file was autogenerated by accel_lut.jl.\n"
    comment_body2 = "//DO NOT MANUALLY EDIT THIS FILE!!!\n\n"
    comment_body3 = "//This file implements accel_lut rom for lookup of quadrilateral corne
    comment_tail = "///////////////////////////////////////////////////////////////////////
    code_preamble1 = "module accel_lut(input clk, input[11:0] accel_val, output reg[75:0] q
    code_preamble2 = "always @(posedge clk) begin\n"
    code_preamble3 = "\tcase (accel_val)\n";
    fs = open(path, "w")
    write(fs, string(comment_head, comment_body1, comment_body2, comment_body3, comment_tai

    # write body
    for i=0:2^12-1
        val = quad_corners[i+1]
        line_str = string("\t\t12'd", i, ": quad_corners = 76'd", val, ";\n")
        write(fs, line_str)
    end

    # write footer
    code_end = "\tendcase\nend\nendmodule\n"
    write(fs, code_end)
    close(fs)
end

function accel_lut(in_path, out_path)
    x_accel, y_accel, x1, y1, x2, y2, x3, y3, x4, y4 = parse_data(in_path)
    write_file(out_path, x_accel, y_accel, x1, y1, x2, y2, x3, y3, x4, y4)
end
```

A.3.4 accel_lut.txt

```
20,20,      0,0,        0,1df,      27f,1df,        27f,0
20,24,      a,d,        0,1df,      27f,1df,        275,d
20,28,      12,1a,      0,1df,      27f,1df,        26e,1a
20,2c,      1b,39,      0,1df,      27f,1df,        265,39
20,30,      1f,43,      0,1df,      27f,1df,        263,43
20,34,      27,4d,      0,1df,      27f,1df,        25b,4d
28,20,      28,0,       0,1b9,      25c,1df,        27f,1e
30,20,      5a,0,       0,176,      22c,1df,        27f,72
34,30,      6e,0,       16,155,     20f,1df,        26a,86
18,20,      0,33,       3b,1df,     27f,1a5,        249,0
10,20,      0,07e,      60,1df,     27f,158,        21b,0
20,18,      8,0,        16,1df,     26a,1df,        27f,0
```

A.3.5 pixels_kept.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
///////////////////////////////////////////////////////////////////////////////////////////
// pixels_kept: Calculates the percentage of pixels lost, given the
// coordinates of the four points of the quadrilateral.
// The module is a pure combinational logic module
// The area formula is given by a standard determinant expansion, and may be
// derived easily.
// Alternatively, it is easily available on the web
///////////////////////////////////////////////////////////////////////////////////////////

module pixels_kept(input[9:0] x1,
                input[8:0] y1,
                input[9:0] x2,
                input[8:0] y2,
                input[9:0] x3,
                input[8:0] y3,
                input[9:0] x4,
                input[8:0] y4,
                output wire[6:0] percent_kept); // percent_kept ranges from 0 to 100, a 7 b

wire signed[10:0] sx1, sx2, sx3, sx4;
wire signed[9:0] sy1, sy2, sy3, sy4;
wire signed[10:0] d_x1_x3, d_x2_x4;
wire signed[9:0] d_y1_y3, d_y2_y4;
wire signed[20:0] prod0, prod1;
wire signed[20:0] prod;
wire signed[20:0] abs_prod;
wire[20:0] unsigned_prod;
wire[13:0] shift_prod_7;
wire[11:0] shift_prod_9;
wire[9:0] shift_prod_11;
wire[14:0] sum_shift_prod;

// sign extensions
assign sx1 = {1'b0, x1};
assign sx2 = {1'b0, x2};
assign sx3 = {1'b0, x3};
assign sx4 = {1'b0, x4};
assign sy1 = {1'b0, y1};
assign sy2 = {1'b0, y2};
assign sy3 = {1'b0, y3};
assign sy4 = {1'b0, y4};

// difference terms
assign d_x1_x3 = sx1 - sx3;
assign d_x2_x4 = sx2 - sx4;
assign d_y1_y3 = sy1 - sy3;
assign d_y2_y4 = sy2 - sy4;

// multipliers
assign prod0 = d_x1_x3 * d_y2_y4;
assign prod1 = d_y1_y3 * d_x2_x4;

// final area calculation
assign prod = prod0 - prod1; // this is twice the area

// but first, we need to take its absolute value
assign abs_prod = (prod < 0) ? -prod : prod;
assign unsigned_prod = abs_prod;

// to compute the percentage of pixels covered, here is the calculation
// we want (100*A)/(640*480), or A/(64*48)
// what we have is temp=2*A
// thus, we need temp/(128*48) = temp/(6144) = temp/(2^11 * 3) = (temp >> 11) / 3
// to avoid the division by 3, we approximate 3 ~= 21/64 (accurate to
// within 1%)
// thus, we want ((temp >> 11)*21) >> 6
// but mult by 21 is same as mult by (16 + 4 + 1)
// thus, our final calculation is ((temp >> 7) + (temp >> 9) + (temp >> 11))>>6
assign shift_prod_7 = unsigned_prod >> 7;
assign shift_prod_9 = unsigned_prod >> 9;
assign shift_prod_11 = unsigned_prod >> 11;
assign sum_shift_prod = shift_prod_7 + shift_prod_9 + shift_prod_11;
assign percent_kept = sum_shift_prod >> 6;

endmodule
```

A.3.6 bram.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
```

```
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
‘default_nettype none
////////////////////////////////////////////////////////////////////////////////////////////
// A simple true dual-port bram module, with hardcoded sizes
// number of lines: 320*240 = 76800
// data word width: 12 bits (4 bits r, 4 bits g, 4 bits b, one pixel per line)
// use here is to store a (downsampled) 640x480 frame at reduced resolution
// that can fit in bram (approx 1 Mbit usage per instantiation)
// Xilinx ISE infers the correct synthesis, and thus this module avoids
// unnecessary Coregen usage
//
// credits: http://danstrother.com/2010/09/11/inferring-rams-in-fpgas/
////////////////////////////////////////////////////////////////////////////////////////////
module bram(input wire a_clk,
    input wire a_wr,
    input wire[16:0] a_addr,
    input wire[11:0] a_din,
    input wire b_clk,
    input wire[16:0] b_addr,
    output reg[11:0] b_dout);

// Shared memory
reg[11:0] mem[76799:0];

// Port A
always @(posedge a_clk) begin
    if (a_wr) begin
        mem[a_addr] <= a_din;
    end
end

// Port B
always @(posedge b_clk) begin
    b_dout <= mem[b_addr];
end

endmodule
```

A.3.7 addr_map.v

```
/*
Copyright (C) {2014}  {James Thomas} <jamesjoethomas@gmail.com>
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
///////////////////////////////////////////////////////////////////////////////////////////
// a simple module for mapping hcount and vcount to address in bram
// the math:
// bram is 320*240 = 76800 lines, 320 columns, and 240 rows
// each line of bram corresponds to one pixel
// currently, each line is 12 bits (4 pixels r, 4 pixels g, 4 pixels b)
// hcount and vcount are in the 640x480 space
// Thus, the desired address is: 320*(vcount/2) + (hcount/2)
// = (128 + 32)vcount + hcount/2
///////////////////////////////////////////////////////////////////////////////////////////

module addr_map(input[9:0] hcount,
                input[9:0] vcount,
                output[16:0] addr);

assign addr = (vcount[9:1] << 8) + (vcount[9:1] << 6) + (hcount >> 1);
endmodule
```

A.3.8 slow_clk.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
```

```

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
//////////////////////////////////////////////////////////////////////////////////////////////
// this module generates a VERY SLOW clk by a simple counter
// note: this method is NOT robust to timing issues, and for slowing
// down/speeding up a clk by a reasonable multiple (e.g 2, 3), use DCM instead
// to guarantee phase locking, elimination of most skew, etc
// Here, the intent is only to generate a pulse with a time period of order of
// seconds
//////////////////////////////////////////////////////////////////////////////////////////////
module slow_clk(input clk, output slow_clk);
    parameter TICKS = 27'd49_999_999;

    reg [31:0] count = 0;
    reg sig_reg = 0;

    always @(posedge clk) begin
        if (count == TICKS) begin
            // flip at half period
            sig_reg <= ~sig_reg;
            count <= 0;
        end
        else begin
            count <= count + 1;
        end
    end
    assign slow_clk = sig_reg;
endmodule
```

A.3.9 move_cursor.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

```

```verilog
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
///////////////////////////////////////////////////////////////////////////////////////////
// move_cursor: This module implements a simple UI for manually adjusting the
// projector correction via pressing the arrow keys, and selecting which
// corner of the quadrilateral the user is manipulating via switch[1:0] positions.
// 00 -> point 1, 01 -> point 2, 10 -> point 3, 11 -> point 4
// All the adjustments can only happen when the override is pressed.
// Inputs are xi_raw, yi_raw (obtained from accelerometer lut)
// Outputs are xi, yi and display_x, display_y (for hex display).
// The intention is to run this on a slow clk, even vsync could be a little
// too fast
///////////////////////////////////////////////////////////////////////////////////////////
module move_cursor(input clk,
                input up,
                input down,
                input left,
                input right,
                input override,
                input[1:0] switch,
                input[9:0] x1_raw,
                input[8:0] y1_raw,
                input[9:0] x2_raw,
                input[8:0] y2_raw,
                input[9:0] x3_raw,
                input[8:0] y3_raw,
                input[9:0] x4_raw,
                input[8:0] y4_raw,
                output reg[9:0] x1,
                output reg[8:0] y1,
                output reg[9:0] x2,
                output reg[8:0] y2,
                output reg[9:0] x3,
                output reg[8:0] y3,
                output reg[9:0] x4,
                output reg[8:0] y4,
                output reg[9:0] display_x,
                output reg[8:0] display_y);

parameter OVERRIDE = 1'b0;

parameter XSPEED = 1’d1;
parameter YSPEED = 1’d1;

// 640 x 480 screen
parameter SCR_WIDTH = 10’d639;
parameter SCR_HEIGHT = 9’d479;

reg cur_state = ~OVERRIDE;

always @(posedge clk) begin
    case (switch)
        2’b00: begin
            display_x <= x1;
            display_y <= y1;
        end
        2’b01: begin
            display_x <= x2;
            display_y <= y2;
        end
        2’b10: begin
            display_x <= x3;
            display_y <= y3;
        end
        2’b11: begin
            display_x <= x4;
            display_y <= y4;
        end
    endcase
end

always @(posedge clk) begin
    if (override && !(cur_state == OVERRIDE)) begin
        cur_state <= OVERRIDE;
        x1 <= x1_raw;
        y1 <= y1_raw;
        x2 <= x2_raw;
        y2 <= y2_raw;
        x3 <= x3_raw;
        y3 <= y3_raw;
        x4 <= x4_raw;
        y4 <= y4_raw;
    end
    else if (override) begin
        case (switch)
            2’b00: begin
```

```
                if (down) begin
                    y1 <= (y1 <= SCR_HEIGHT-YSPEED) ? (y1 + YSPEED) : y1;
                end
                else if (up) begin
                    y1 <= (y1 >= YSPEED) ? (y1 - YSPEED) : y1;
                end
                else if (left) begin
                    x1 <= (x1 >= XSPEED) ? (x1 - XSPEED) : x1;
                end
                else if (right) begin
                    x1 <= (x1 <= SCR_WIDTH-XSPEED) ? (x1 + XSPEED) : x1;
                end
            end
            2'b01: begin
                if (down) begin
                    y2 <= (y2 <= SCR_HEIGHT-YSPEED) ? (y2 + YSPEED) : y2;
                end
                else if (up) begin
                    y2 <= (y2 >= YSPEED) ? (y2 - YSPEED) : y2;
                end
                else if (left) begin
                    x2 <= (x2 >= XSPEED) ? (x2 - XSPEED) : x2;
                end
                else if (right) begin
                    x2 <= (x2 <= SCR_WIDTH-XSPEED) ? (x2 + XSPEED) : x2;
                end
            end
            2'b10: begin
                if (down) begin
                    y3 <= (y3 <= SCR_HEIGHT-YSPEED) ? (y3 + YSPEED) : y3;
                end
                else if (up) begin
                    y3 <= (y3 >= YSPEED) ? (y3 - YSPEED) : y3;
                end
                else if (left) begin
                    x3 <= (x3 >= XSPEED) ? (x3 - XSPEED) : x3;
                end
                else if (right) begin
                    x3 <= (x3 <= SCR_WIDTH-XSPEED) ? (x3 + XSPEED) : x3;
                end
            end
            2'b11: begin
                if (down) begin
                    y4 <= (y4 <= SCR_HEIGHT-YSPEED) ? (y4 + YSPEED) : y4;
                end
                else if (up) begin
```

```
                    y4 <= (y4 >= YSPEED) ? (y4 - YSPEED) : y4;
                end
                else if (left) begin
                    x4 <= (x4 >= XSPEED) ? (x4 - XSPEED) : x4;
                end
                else if (right) begin
                    x4 <= (x4 <= SCR_WIDTH-XSPEED) ? (x4 + XSPEED) : x4;
                end
            end
        endcase
    end
    else begin
        x1 <= x1_raw;
        y1 <= y1_raw;
        x2 <= x2_raw;
        y2 <= y2_raw;
        x3 <= x3_raw;
        y3 <= y3_raw;
        x4 <= x4_raw;
        y4 <= y4_raw;
        cur_state <= ~OVERRIDE;
    end
end

endmodule
```

A.3.10 perspective_params.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
///////////////////////////////////////////////////////////////////////////////////////////
// perspective_params: Generate the parameters for the perspective transform from the
// rectangle to the quadrilateral inside it
// Note that this is the forward mapping
// The math is described as follows
// Let (x1, y1), (x2, y2), (x3, y3), (x4, y4) be the four points inside the
// screen
// Let the (forward) perspective map be given by:
// (X, Y) = ((p1*x + p2*y + p3)/(p7*x + p8*y + p9), (p4*x + p5*y + p6)/(p7*x
// + p8*y + p9))
// Then our task is to determine the values of p_i given the values of the x_i
// This is a system of equations in 8 unknowns
// It turns out that a pretty simple closed form solution exists, given by
//
// p7 = 3((x1-x4)(y2-y3) + 3(y1-y4)(x3-x2))
// p8 = 4((x1-x2)(y3-y4) + 4(x4-x3)(y1-y2))
// denom = x4(y2-y3) + x2(y3-y4) + x3(y4-y2)
// p9 = 1920*denom (2^7 * 15 * denom)
// p3 = 1920*x1*denom (2^7 * 15 * x1 * denom)
// p6 = 1920*y1*denom (2^7 * 15 * y1 * denom)
// p1 = x4*p7 + 3(x4-x1)*denom
// p2 = x2*p8 + 4(x2-x1)*denom
// p4 = y4*p7 + 3(y4-y1)*denom
// p5 = y2*p8 + 4(y2-y1)*denom
//
// inverse mapping
// p1_inv = p6*p8 - p5*p9
// p2_inv = p2*p9 - p3*p8
// p3_inv = p3*p5 - p2*p6
// p4_inv = p4*p9 - p6*p7
// p5_inv = p3*p7 - p1*p9
// p6_inv = p1*p6 - p3*p4
// p7_inv = p5*p7 - p4*p8
// p8_inv = p1*p8 - p2*p7
// p9_inv = p2*p4 - p1*p5
// dec_numx_horiz = p1_inv * 639
// dec_numy_horiz = p4_inv * 639
// dec_denom_horiz = p7_inv * 639
//
// Future improvements:
// 1)
// This module uses over 120 out of 144 available 18x18
// multipliers!!!
// By reducing bitwidths and avoiding needless multiplies, e.g shifting
// whenever multiplying by constant, resource utilization could be improved
// Even with those improvements, I estimate the need of at least 80-100 18x18
// multipliers to avoid precision loss
//
```

```verilog
// 2)
// Right now, the intention is to run this module on a slow clock, since we
// don't want the parameters to change mid-frame anyway.
// Thus, timing is never an issue right now.
// However, module is easily pipelined, if one needs to run at fast clock.
///////////////////////////////////////////////////////////////////////////////////////////

module perspective_params(input clk,
                input[9:0] x1,
                input[8:0] y1,
                input[9:0] x2,
                input[8:0] y2,
                input[9:0] x3,
                input[8:0] y3,
                input[9:0] x4,
                input[8:0] y4,
                // reason for the hardcoded numbers is FPGA limitations on
                // multiplier bitwidths (s18 x s18 yields s35)
                // Note: guaranteed, mathematically proven bitwidths are:
                // forward: 36, 36, 44, 35, 35, 43, 24, 24, 33
                // inverse: 68, 69, 79, 68, 69, 79, 59, 60, 71
                output reg signed[67:0] p1_inv,
                output reg signed[68:0] p2_inv,
                output reg signed[78:0] p3_inv,
                output reg signed[67:0] p4_inv,
                output reg signed[68:0] p5_inv,
                output reg signed[78:0] p6_inv,
                output reg signed[58:0] p7_inv,
                output reg signed[59:0] p8_inv,
                output reg signed[70:0] p9_inv,
                output reg signed[78:0] dec_numx_horiz,
                output reg signed[78:0] dec_numy_horiz,
                output reg signed[70:0] dec_denom_horiz);

// sign extensions
wire signed[10:0] sx1, sx2, sx3, sx4;
wire signed[9:0] sy1, sy2, sy3, sy4;
assign sx1 = {1'b0, x1};
assign sx2 = {1'b0, x2};
assign sx3 = {1'b0, x3};
assign sx4 = {1'b0, x4};
assign sy1 = {1'b0, y1};
assign sy2 = {1'b0, y2};
assign sy3 = {1'b0, y3};
assign sy4 = {1'b0, y4};

```

```
// difference values for computation
wire signed[10:0] d_x1_x2,d_x2_x3,d_x3_x4,d_x4_x1;
wire signed[9:0] d_y1_y2, d_y2_y3, d_y3_y4, d_y4_y1, d_y4_y2;
assign d_x1_x2 = sx1 - sx2;
assign d_x2_x3 = sx2 - sx3;
assign d_x3_x4 = sx3 - sx4;
assign d_x4_x1 = sx4 - sx1;
assign d_y1_y2 = sy1 - sy2;
assign d_y2_y3 = sy2 - sy3;
assign d_y3_y4 = sy3 - sy4;
assign d_y4_y1 = sy4 - sy1;
assign d_y4_y2 = sy4 - sy2;

// computation of p7, p8
wire signed[20:0] num0, num1, num2, num3;
wire signed[21:0] p7_temp, p8_temp;
wire signed[23:0] p7, p8;
assign num0 = -(d_x4_x1 * d_y2_y3);
assign num1 = d_y4_y1 * d_x2_x3;
assign num2 = d_x1_x2 * d_y3_y4;
assign num3 = -(d_x3_x4 * d_y1_y2);
assign p7_temp = num0 + num1;
assign p8_temp = num2 + num3;
assign p7 = (p7_temp <<< 1) + p7_temp;
assign p8 = (p8_temp <<< 2);

// computation of denom
wire signed[20:0] denom0, denom1, denom2;
wire signed[21:0] denom;
assign denom0 = sx4 * d_y2_y3;
assign denom1 = sx2 * d_y3_y4;
assign denom2 = sx3 * d_y4_y2;
assign denom = denom0 + denom1 + denom2;

// computation of p3, p6, p9
// observe that 1920 = 2^7 * 15
wire signed[25:0] denom_15;
wire signed[32:0] p9;
wire signed[32:0] x1_denom;
wire signed[36:0] x1_denom_15;
wire signed[43:0] p3;
wire signed[31:0] y1_denom;
wire signed[35:0] y1_denom_15;
wire signed[42:0] p6;
assign denom_15 = (denom <<< 4) - denom; // denom * 15
assign p9 = denom_15 <<< 7; // denom * 1920
assign x1_denom = sx1 * denom; // x1 * denom
assign x1_denom_15 = (x1_denom <<< 4) - x1_denom; // x1 * denom * 15
assign p3 = x1_denom_15 <<< 7; // x1 * denom * 1920
assign y1_denom = sy1 * denom; // y1 * denom
assign y1_denom_15 = (y1_denom <<< 4) - y1_denom; // y1 * denom * 15
assign p6 = y1_denom_15 <<< 7; // y1 * denom * 1920

// computation of p1, p2, p4, p5
wire signed[32:0] d_x1_x2_denom;
wire signed[32:0] d_x4_x1_denom;
wire signed[31:0] d_y4_y1_denom;
wire signed[31:0] d_y1_y2_denom;
wire signed[34:0] d_x1_x2_denom_scale;
wire signed[34:0] d_x4_x1_denom_scale;
wire signed[33:0] d_y4_y1_denom_scale;
wire signed[33:0] d_y1_y2_denom_scale;
wire signed[34:0] x4_p7;
wire signed[34:0] x2_p8;
wire signed[33:0] y4_p7;
wire signed[33:0] y2_p8;
wire signed[35:0] p1, p2;
wire signed[34:0] p4, p5;
assign d_x1_x2_denom = d_x1_x2 * denom;
assign d_x4_x1_denom = d_x4_x1 * denom;
assign d_y4_y1_denom = d_y4_y1 * denom;
assign d_y1_y2_denom = d_y1_y2 * denom;
assign d_x4_x1_denom_scale = (d_x4_x1_denom <<< 1) + d_x4_x1_denom; // d_x4_x1_denom*3
assign d_x1_x2_denom_scale = (d_x1_x2_denom <<< 2); // d_x1_x2_denom*4
assign d_y4_y1_denom_scale = (d_y4_y1_denom <<< 1) + d_y4_y1_denom; // d_y4_y1_denom*3
assign d_y1_y2_denom_scale = (d_y1_y2_denom <<< 2); // d_y1_y2_denom*4
assign x4_p7 = sx4 * p7;
assign x2_p8 = sx2 * p8;
assign y4_p7 = sy4 * p7;
assign y2_p8 = sy2 * p8;
assign p1 = x4_p7 + d_x4_x1_denom_scale;
assign p2 = x2_p8 - d_x1_x2_denom_scale;
assign p4 = y4_p7 + d_y4_y1_denom_scale;
assign p5 = y2_p8 - d_y1_y2_denom_scale;

// 36, 36, 44, 35, 35, 43, 24, 24, 33
// computation of inverse mapping
wire signed[67:0] p1_inv_wire;
wire signed[68:0] p2_inv_wire;
wire signed[78:0] p3_inv_wire;
wire signed[67:0] p4_inv_wire;
wire signed[68:0] p5_inv_wire;
wire signed[78:0] p6_inv_wire;
wire signed[58:0] p7_inv_wire;
wire signed[59:0] p8_inv_wire;
wire signed[70:0] p9_inv_wire;
assign p1_inv_wire = p6*p8 - p5*p9;
assign p2_inv_wire = p2*p9 - p3*p8;
assign p3_inv_wire = p3*p5 - p2*p6;
assign p4_inv_wire = p4*p9 - p6*p7;
assign p5_inv_wire = p3*p7 - p1*p9;
assign p6_inv_wire = p1*p6 - p3*p4;
assign p7_inv_wire = p5*p7 - p4*p8;
assign p8_inv_wire = p1*p8 - p2*p7;
assign p9_inv_wire = p2*p4 - p1*p5;

// computation of dec_numx_horiz, dec_numy_horiz, dec_denom_horiz
wire signed[78:0] dec_numx_horiz_wire;
wire signed[78:0] dec_numy_horiz_wire;
wire signed[70:0] dec_denom_horiz_wire;
// multiply stuff by 639 = 512 + 128 - 1
assign dec_numx_horiz_wire = (p1_inv_wire <<< 9) + (p1_inv_wire <<< 7) - p1_inv_wire;
assign dec_numy_horiz_wire = (p4_inv_wire <<< 9) + (p4_inv_wire <<< 7) - p4_inv_wire;
assign dec_denom_horiz_wire = (p7_inv_wire <<< 9) + (p7_inv_wire <<< 7) - p7_inv_wire;

always @(posedge clk) begin
    p1_inv <= p1_inv_wire;
    p2_inv <= p2_inv_wire;
    p3_inv <= p3_inv_wire;
    p4_inv <= p4_inv_wire;
    p5_inv <= p5_inv_wire;
    p6_inv <= p6_inv_wire;
    p7_inv <= p7_inv_wire;
    p8_inv <= p8_inv_wire;
    p9_inv <= p9_inv_wire;
    dec_numx_horiz <= dec_numx_horiz_wire;
    dec_numy_horiz <= dec_numy_horiz_wire;
    dec_denom_horiz <= dec_denom_horiz_wire;
end

endmodule
```

A.3.11 pixel_map.v

```
/*
Copyright (C) {2014}  {Ganesh Ajjanagadde} <gajjanagadde@gmail.com>

This program is free software: you can redistribute it and/or modify
```

```
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
//////////////////////////////////////////////////////////////////////////////////////////
// pixel_map: This module performs the core perspective transformation
// It computes (X, Y) = ((p1*x+p2*y+p3)/(p7*x+p8*y+p9),
// (p4*x+p5*y+p6)/(p7*x+p8*y+p9)) given values pi (computed in
// perspective_params.v)
// The module also does the necessary pixel read form ntsc_buf, and writes the
// output to vga_buf
//
// Future work:
// 1) Note the huge bit width of the divider. This results in a ridiculous ~80
// clock cycles per pixel. Pipelining of 80 bit divider can't be done in
// coregen, and will need to be done manually. It would be a nice feature,
// since this would enable a real time projector display as opposed to current
// ~1-2 frames per second
//
// 2) reduce bit widths: these bit widths are conservative, and mathematically
// guaranteed never to lose precision. Software indicates that I can lose up
// to 20 bits of precision, and still be ok. A careful analysis of this needs
// to be performed
//////////////////////////////////////////////////////////////////////////////////////////
`default_nettype none
module pixel_map(input clk,
                input signed[67:0] p1_inv,
                input signed[68:0] p2_inv,
                input signed[78:0] p3_inv,
                input signed[67:0] p4_inv,
                input signed[68:0] p5_inv,
                input signed[78:0] p6_inv,
                input signed[58:0] p7_inv,
                input signed[59:0] p8_inv,
                input signed[70:0] p9_inv,
                input signed[78:0] dec_numx_horiz,
                input signed[78:0] dec_numy_horiz,
                input signed[70:0] dec_denom_horiz,
                input[11:0] pixel_in,
                output reg[11:0] pixel_out,
                output[16:0] ntsc_out_addr,
                output reg vga_in_wr,
                output[16:0] vga_in_addr);

// internal registers for numerator and denominator computation
// see perspective_params.v for the equations
reg signed[78:0] num_x = 0;
reg signed[78:0] num_y = 0;
reg signed[78:0] denom = 0;

// internal registers for pixel index
reg[9:0] cur_x = 0;
reg[9:0] cur_y = 0;

// divider outputs
wire signed[78:0] inv_x;
wire signed[78:0] inv_y;
wire signed[78:0] dummy_remx;
wire signed[78:0] dummy_remy;
reg div_start;
wire div_done_x;
wire div_done_y;

// instantiate dividers
divider #(.WIDTH(79)) divider_x(.clk(clk),
                                .sign(1'b1),
                                .start(div_start),
                                .dividend(num_x),
                                .divider(denom),
                                .quotient(inv_x),
                                .remainder(dummy_remx),
                                .ready(div_done_x));

divider #(.WIDTH(79)) divider_y(.clk(clk),
                                .sign(1'b1),
                                .start(div_start),
                                .dividend(num_y),
                                .divider(denom),
                                .quotient(inv_y),
                                .remainder(dummy_remy),
                                .ready(div_done_y));

// instantiate an address mapper (for the vga_in)
addr_map addr_map_vga(.hcount(cur_x),
                .vcount(cur_y),
                .addr(vga_in_addr));

// instantiate an address mapper (for the ntsc_out)
addr_map addr_map_ntsc(.hcount(inv_x[9:0]),
                    .vcount(inv_y[9:0]),
                    .addr(ntsc_out_addr));

parameter NEXT_PIXEL_ST = 2'b00;
parameter WAIT_FOR_DIV_ST = 2'b01;
parameter WAIT_FOR_MEM_ST = 2'b10;
parameter BLACK = 12'd0;
reg[1:0] cur_state = NEXT_PIXEL_ST;
always @(posedge clk) begin
    case (cur_state)
        NEXT_PIXEL_ST: begin
            vga_in_wr <= 0;
            div_start <= 1;
            cur_state <= WAIT_FOR_DIV_ST;
            if ((cur_x == 639) && (cur_y == 479)) begin
                cur_x <= 0;
                cur_y <= 0;
                num_x <= p3_inv;
                num_y <= p6_inv;
                denom <= p9_inv;
            end
            else if ((cur_x == 639) && (cur_y !=  479)) begin
                cur_x <= 0;
                cur_y <= cur_y + 1;
                num_x <= num_x - dec_numx_horiz + p2_inv;
                num_y <= num_y - dec_numy_horiz + p5_inv;
                denom <= denom - dec_denom_horiz + p8_inv;
            end
            else if (cur_x != 639) begin
                cur_x <= cur_x + 1;
                cur_y <= cur_y;
                num_x <= num_x + p1_inv;
                num_y <= num_y + p4_inv;
                denom <= denom + p7_inv;
            end
        end

        WAIT_FOR_DIV_ST: begin
           vga_in_wr <= 0;
           div_start <= 0;
           if (div_done_x == 1) begin
```

```
               cur_state <= WAIT_FOR_MEM_ST;
           end
        end

        WAIT_FOR_MEM_ST: begin
            if ((inv_x < 0) || (inv_x > 639) || (inv_y < 0) || (inv_y > 479)) begin
                pixel_out <= BLACK;
                vga_in_wr <= 1;
                cur_state <= NEXT_PIXEL_ST;
            end
            else begin
                pixel_out <= pixel_in;
                vga_in_wr <= 1;
                cur_state <= NEXT_PIXEL_ST;
            end
        end
    endcase
end
endmodule
```

A.3.12 audioManager.v

```
/*
Copyright (C) {2014}  {Shawn Jain} <shawnjain.08@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
`default_nettype none
// Receives audio samples via FTDI UM245R USB-to-FIFO, stores to
// onboard flash memory. Plays back the percentage of pixels used
// when audioTrigger is pulsed, where the percent is set by
// audioSelector. Internally it queues upto four separate tracks
// to be played. To save memory and for a faster transfer, the
// system constructs all numbers 1-100 out of a subset of the
// digits.
```

```

module audioManager(
  input wire clock, // 27mhz system clock
  input wire reset, // 1 to reset to initial state

  // User I/O
  input wire startSwitch,
  input wire [6:0] audioSelector,
  input wire writeSwitch,  // 1=Write, 0=Read
  output wire [63:0] hexdisp,
  input wire audioTrigger, // 1=Begin Playback as determined by audioSelector

  // AC97 I/O
  input wire ready,                // 1 when AC97 data is available
  input wire [7:0] from_ac97_data, // 8-bit PCM data from mic
  output reg [7:0] to_ac97_data,   // 8-bit PCM data to headphone

  // Flash I/O
  output wire [15:0] flash_data,
  output wire [23:0] flash_address,
  output wire flash_ce_b,
  output wire flash_oe_b,
  output wire flash_we_b,
  output wire flash_reset_b,
  output wire flash_byte_b,
  input wire flash_sts,
  output wire busy,

  // USB I/O
  input wire [7:0] data, // the data pins from the USB fifo
  input wire rxf,        // the rxf pin from the USB fifo
  output wire rd         // the rd pin from the USB fifo (OUTPUT)
);

  // Playback addresses:
  parameter TRACK_LENGTH = 69000; // approx 1 sec

  parameter ONE_INDEX = 23'd0;
  parameter TWO_INDEX = 23'd1;
  parameter THREE_INDEX = 23'd2;
  parameter FOUR_INDEX = 23'd3;
  parameter FIVE_INDEX = 23'd4;
  parameter SIX_INDEX = 23'd5;
  parameter SEVEN_INDEX = 23'd6;
  parameter EIGHT_INDEX = 23'd7;
  parameter NINE_INDEX = 23'd8;
  parameter TEN_INDEX = 23’d9;
  parameter ELEVEN_INDEX = 23’d10;    // A
  parameter TWELVE_INDEX = 23’d11;    // B
  parameter THIRTEEN_INDEX = 23’d12;  // C
  parameter FOURTEEN_INDEX = 23’d13;  // D
  parameter FIFTEEN_INDEX = 23’d14;   // E
  parameter TWENTY_INDEX = 23’d15;    // F
  parameter THIRTY_INDEX = 23’d16;    // 10
  parameter FOURTY_INDEX = 23’d17;    // 11
  parameter FIFTY_INDEX = 23’d18;     // 12
  parameter SIXTY_INDEX = 23’d19;     // 13
  parameter SEVENTY_INDEX = 23’d20;   // 14
  parameter EIGHTY_INDEX = 23’d21;    // 15
  parameter NINETY_INDEX = 23’d22;    // 16
  parameter HUNDRED_INDEX = 23’d23;   // 17
  parameter TEEN_INDEX = 23’d24;      // 18
  parameter PERCENT_INDEX = 23’d25;   // 19
  parameter USED_INDEX = 23’d26;      // 1A
  parameter HELP_AUDIO_INDEX = 23’d27;// 1B
  parameter SKIP_INDEX = 23’d28;      // 1C
  parameter UNUSED_INDEX = 23’d31;    // 1F

  reg writemode = 0;         // 1=write mode; 0=read mode
  reg [15:0] wdata = 0;      // writeData
  reg dowrite = 0;           // 1=new data, write it
  reg [22:0] raddr = 2;      // readAddress
  wire [15:0] frdata;        // readData
  reg doread = 0;            // 1=execute read

  flash_manager fm(
    .clock(clock),
    .reset(reset),

    // Interface I/O
    .writemode(writemode),
    .wdata(wdata),
    .dowrite(dowrite),
    .raddr(raddr),
    .frdata(frdata),
    .doread(doread),
    .busy(busy),

    // Flash I/O
    .flash_data(flash_data),
    .flash_address(flash_address),
    .flash_ce_b(flash_ce_b),
    .flash_oe_b(flash_oe_b),
    .flash_we_b(flash_we_b),
    .flash_reset_b(flash_reset_b),
    .flash_sts(flash_sts),
    .flash_byte_b(flash_byte_b)
  );

  wire [7:0] out;// data from FIFO (OUTPUT)
  wire newout;   // newout=1 out contains new data (OUTPUT)
  wire hold;     // hold=1 the module will not accept new data from the FIFO

  assign hold = 1'b0;

  usb_input usbtest(
    .clk(clock),
    .reset(reset),

    // USB FTDI I/O
    .data(data[7:0]),
    .rxf(rxf),
    .rd(rd),

    // Interface
    .out(out[7:0]),
    .newout(newout),
    .hold(hold)
  );

  wire [3:0] hundreds;
  wire [3:0] tens;
  wire [3:0] ones;

  BCD inputToBCD(
    .number({1'b0, audioSelector}),
    .hundreds(hundreds),
    .tens(tens),
    .ones(ones)
  );

  reg lastAudioTrigger;
  reg [2:0] third = 0;
  reg lastReady;

  // Set of 4 addresses that represent a playback sequence
  // First track in bottom 23 bits[22:0]. Last track in top bits [91:68].
  reg [91:0] playbackSeq = 2;
```

```
reg [22:0] trackEndAddr = 0;
reg playing = 0;
reg lastPlaying = 0;
reg [15:0] bytesRxed = 0;

assign hexdisp = {playbackSeq[30:23], playbackSeq[7:0], 1'h0 ,trackEndAddr, 1'h0, raddr[22

reg [7:0] dataFromFifo;
always @ (posedge rd) begin
  dataFromFifo <= out; // out & data have same results
end

always @ (posedge clock) begin
  lastAudioTrigger <= audioTrigger;
  lastReady <= ready;
  lastPlaying <= playing;

  if (startSwitch) begin
    // write USB RX data if switch is up
    if (writeSwitch) begin
      writemode <= 1'b1;
      doread <= 1'b0;
      //dowrite <= 1'b0; // only write on new data // WATCH OUT!!
      if (newout) begin
        bytesRxed <= bytesRxed + 1;
        wdata <= {dataFromFifo, 8'b0};//{out, 8'b0};
        dowrite <= 1'b1;
      end
    end

    // if button is DOWN - scroll through addresses via buttons
    if (~writeSwitch) begin
      dowrite <= 1'b0;
      writemode <= 1'b0;
      doread <= 1'b1;

      if (playing & ready) begin // REMOVE audioTrigger
        if (raddr < trackEndAddr) begin
          // Normal 48K Playback
          raddr <= raddr + 1;
          to_ac97_data <= frdata[15:8]; // PUT BACK
        end
        else begin
          if (playbackSeq[45:23] < UNUSED_INDEX) begin
            // change raddr to next track
            raddr <= playbackSeq[45:23] * TRACK_LENGTH;
            // shift playbackSeq down
            playbackSeq <= {UNUSED_INDEX, playbackSeq[91:23]};
            // update trackEndAddr
            trackEndAddr <= playbackSeq[45:23] * TRACK_LENGTH + TRACK_LENGTH;
          end
          else if (playbackSeq[45:23] == UNUSED_INDEX) begin
            playing <= 0;
            raddr <= 0; // reset for safety - lower than UNUSED_ADDR
          end
        end
      end // if (playing & audioTrigger & ready)

      // if entering this state, assign start address
      if (audioTrigger & ~lastAudioTrigger) begin
        playing <= 1;
        case(ones)
          0: playbackSeq[91:23] <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX};
          1: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, ONE_INDEX};
          2: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, TWO_INDEX};
          3: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, THREE_INDEX};
          4: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, FOUR_INDEX};
          5: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, FIVE_INDEX};
          6: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, SIX_INDEX};
          7: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, SEVEN_INDEX};
          8: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, EIGHT_INDEX};
          9: playbackSeq[91:23] <= {USED_INDEX, PERCENT_INDEX, NINE_INDEX};
          default:  playbackSeq <= {USED_INDEX, PERCENT_INDEX, UNUSED_INDEX}; // error
        endcase
        case (tens)
          0: playbackSeq[22:0] <= SKIP_INDEX;
          1: playbackSeq[22:0] <= TEN_INDEX;
          2: playbackSeq[22:0] <= TWENTY_INDEX;
          3: playbackSeq[22:0] <= THIRTY_INDEX;
          4: playbackSeq[22:0] <= FOURTY_INDEX;
          5: playbackSeq[22:0] <= FIFTY_INDEX;
          6: playbackSeq[22:0] <= SIXTY_INDEX;
          7: playbackSeq[22:0] <= SEVENTY_INDEX;
          8: playbackSeq[22:0] <= EIGHTY_INDEX;
          9: playbackSeq[22:0] <= NINETY_INDEX;
          default: playbackSeq[22:0] <= UNUSED_INDEX;
        endcase
        case (hundreds)
          0: begin end
          1: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, HUNDRED_INDEX}; //
        endcase
        case (audioSelector)
      11: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, ELEVEN_INDEX};
      12: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, TWELVE_INDEX};
      13: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, THIRTEEN_INDEX};
      14: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, FOURTEEN_INDEX};
      15: playbackSeq <= {UNUSED_INDEX, USED_INDEX, PERCENT_INDEX, FIFTEEN_INDEX};
      16: playbackSeq <= {USED_INDEX, PERCENT_INDEX, TEEN_INDEX, SIX_INDEX};
      17: playbackSeq <= {USED_INDEX, PERCENT_INDEX, TEEN_INDEX, SEVEN_INDEX};
      18: playbackSeq <= {USED_INDEX, PERCENT_INDEX, TEEN_INDEX, EIGHT_INDEX};
      19: playbackSeq <= {USED_INDEX, PERCENT_INDEX, TEEN_INDEX, NINE_INDEX};
      default: begin end
     endcase
    end // if (audioTrigger & ~lastAudioTrigger)

    // just started playing - need to set raddr
    // Assuming this happens once playbackSeq has been properly set
    if (playing & ~lastPlaying) begin
     if (playbackSeq[22:0] == SKIP_INDEX) begin
      playbackSeq <= {UNUSED_INDEX, playbackSeq[91:23]};
      raddr <= playbackSeq[45:23] * TRACK_LENGTH;
      trackEndAddr <= playbackSeq[45:23] * TRACK_LENGTH + TRACK_LENGTH;
     end
     else begin
      raddr <= playbackSeq[22:0] * TRACK_LENGTH;
      trackEndAddr <= playbackSeq[22:0] * TRACK_LENGTH + TRACK_LENGTH;
     end
    end
   end // if (~writeSwitch)
  end // if (startSwitch)
  else begin
   // TO ENABLE RESET:
   // writemode <= 1
   // dowrite <= 0
   // doread <= 0 // to be safe

   // Reset First, Write Second, Read Later
   writemode <= 1’h1;
   doread <= 1’h0;
   dowrite <= 1’h0;
  end
 end // always @
endmodule
```

A.3.13 binaryToDecimal.py

```
# Shawn Jain
# Python script to generate LUT for BCD.v
```

```

for i in range(100):
        print str(i) + ’: begin ’ + ’ones <= ’ + str(i%10) + ’; ’ + ’tens <= ’ + str(i/10) +
print ’default: begin ones <= 0; tens <= 0; end’
```

A.3.14 BCD.v

```
/*
Copyright (C) {2014}  {Shawn Jain} <shawnjain.08@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
// Converts an 8-bit binary number into a decimal representation
// LUT generated from assets/binaryToDecimal.py

module BCD(
   input wire [7:0] number,
   output reg [3:0] hundreds,
   output reg [3:0] tens,
   output reg [3:0] ones);

   always @ (number) begin
      case(number)
         0: begin ones <= 0; tens <= 0; end
         1: begin ones <= 1; tens <= 0; end
         2: begin ones <= 2; tens <= 0; end
         3: begin ones <= 3; tens <= 0; end
         4: begin ones <= 4; tens <= 0; end
         5: begin ones <= 5; tens <= 0; end
         6: begin ones <= 6; tens <= 0; end
         7: begin ones <= 7; tens <= 0; end
         8: begin ones <= 8; tens <= 0; end
         9: begin ones <= 9; tens <= 0; end
         10: begin ones <= 0; tens <= 1; end
         11: begin ones <= 1; tens <= 1; end
```

```
            12: begin ones <= 2; tens <= 1; end
            13: begin ones <= 3; tens <= 1; end
            14: begin ones <= 4; tens <= 1; end
            15: begin ones <= 5; tens <= 1; end
            16: begin ones <= 6; tens <= 1; end
            17: begin ones <= 7; tens <= 1; end
            18: begin ones <= 8; tens <= 1; end
            19: begin ones <= 9; tens <= 1; end
            20: begin ones <= 0; tens <= 2; end
            21: begin ones <= 1; tens <= 2; end
            22: begin ones <= 2; tens <= 2; end
            23: begin ones <= 3; tens <= 2; end
            24: begin ones <= 4; tens <= 2; end
            25: begin ones <= 5; tens <= 2; end
            26: begin ones <= 6; tens <= 2; end
            27: begin ones <= 7; tens <= 2; end
            28: begin ones <= 8; tens <= 2; end
            29: begin ones <= 9; tens <= 2; end
            30: begin ones <= 0; tens <= 3; end
            31: begin ones <= 1; tens <= 3; end
            32: begin ones <= 2; tens <= 3; end
            33: begin ones <= 3; tens <= 3; end
            34: begin ones <= 4; tens <= 3; end
            35: begin ones <= 5; tens <= 3; end
            36: begin ones <= 6; tens <= 3; end
            37: begin ones <= 7; tens <= 3; end
            38: begin ones <= 8; tens <= 3; end
            39: begin ones <= 9; tens <= 3; end
            40: begin ones <= 0; tens <= 4; end
            41: begin ones <= 1; tens <= 4; end
            42: begin ones <= 2; tens <= 4; end
            43: begin ones <= 3; tens <= 4; end
            44: begin ones <= 4; tens <= 4; end
            45: begin ones <= 5; tens <= 4; end
            46: begin ones <= 6; tens <= 4; end
            47: begin ones <= 7; tens <= 4; end
            48: begin ones <= 8; tens <= 4; end
            49: begin ones <= 9; tens <= 4; end
            50: begin ones <= 0; tens <= 5; end
            51: begin ones <= 1; tens <= 5; end
            52: begin ones <= 2; tens <= 5; end
            53: begin ones <= 3; tens <= 5; end
            54: begin ones <= 4; tens <= 5; end
            55: begin ones <= 5; tens <= 5; end
            56: begin ones <= 6; tens <= 5; end
            57: begin ones <= 7; tens <= 5; end
```

```
          58: begin ones <= 8; tens <= 5; end
          59: begin ones <= 9; tens <= 5; end
          60: begin ones <= 0; tens <= 6; end
          61: begin ones <= 1; tens <= 6; end
          62: begin ones <= 2; tens <= 6; end
          63: begin ones <= 3; tens <= 6; end
          64: begin ones <= 4; tens <= 6; end
          65: begin ones <= 5; tens <= 6; end
          66: begin ones <= 6; tens <= 6; end
          67: begin ones <= 7; tens <= 6; end
          68: begin ones <= 8; tens <= 6; end
          69: begin ones <= 9; tens <= 6; end
          70: begin ones <= 0; tens <= 7; end
          71: begin ones <= 1; tens <= 7; end
          72: begin ones <= 2; tens <= 7; end
          73: begin ones <= 3; tens <= 7; end
          74: begin ones <= 4; tens <= 7; end
          75: begin ones <= 5; tens <= 7; end
          76: begin ones <= 6; tens <= 7; end
          77: begin ones <= 7; tens <= 7; end
          78: begin ones <= 8; tens <= 7; end
          79: begin ones <= 9; tens <= 7; end
          80: begin ones <= 0; tens <= 8; end
          81: begin ones <= 1; tens <= 8; end
          82: begin ones <= 2; tens <= 8; end
          83: begin ones <= 3; tens <= 8; end
          84: begin ones <= 4; tens <= 8; end
          85: begin ones <= 5; tens <= 8; end
          86: begin ones <= 6; tens <= 8; end
          87: begin ones <= 7; tens <= 8; end
          88: begin ones <= 8; tens <= 8; end
          89: begin ones <= 9; tens <= 8; end
          90: begin ones <= 0; tens <= 9; end
          91: begin ones <= 1; tens <= 9; end
          92: begin ones <= 2; tens <= 9; end
          93: begin ones <= 3; tens <= 9; end
          94: begin ones <= 4; tens <= 9; end
          95: begin ones <= 5; tens <= 9; end
          96: begin ones <= 6; tens <= 9; end
          97: begin ones <= 7; tens <= 9; end
          98: begin ones <= 8; tens <= 9; end
          99: begin ones <= 9; tens <= 9; end
          default: begin ones <= 0; tens <= 0; end
       endcase
       hundreds <= 0;
    end
```

```
endmodule

// Note: a computational logic based binary to BCD is found at:
// http://www.deathbylogic.com/2013/12/binary-to-binary-coded-decimal-bcd-converter/

module BCDTest;
   reg [7:0] number = 8'd65;
   wire [3:0] hundreds;
   wire [3:0] tens;
   wire [3:0] ones;

   BCD bc(number, hundreds, tens, ones);
   initial begin
      #100
      $display("%d, %d, %d", hundreds, tens, ones);
      $stop();
   end
endmodule
```

A.3.15 ClockDivider.v

```
/*
Copyright (C) {2014}  {Shawn Jain} <shawnjain.08@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
// From Lab 4
// Sends a pulse on oneHertz_enable every Hz clock cycles

module ClockDivider #(parameter Hz = 27000000)(
        input clock, reset, fastMode,
        output reg oneHertz_enable
        );

        reg [24:0] counter = 25'b0;

        always @ (posedge clock) begin
                if (reset) begin
                        counter <= 25'b0;
                        oneHertz_enable <= 1'b0;
                end
                else if (counter == (fastMode ? 3:Hz)) begin
                        oneHertz_enable <= 1'b1;
                        counter <= 25'b0;
                end
                else begin
                        counter <= counter + 1;
                        oneHertz_enable <= 1'b0;
                end
        end

endmodule
```

A.3.16 Square.v

```
/*
Copyright (C) {2014}  {Shawn Jain} <shawnjain.08@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
// From Lab 4
// Generates a square wave that flips every Hz clock cycles

module Square #(parameter Hz = 27000000) (
        input clock, reset,
        output reg square = 0);

        wire oneHertz_enable;

        ClockDivider #(.Hz(Hz)) Sqr (
                .clock(clock),
                .reset(reset),
                .fastMode(1'b0),
                .oneHertz_enable(oneHertz_enable)
        );

        always @ (posedge oneHertz_enable) begin
                square <= ~square;
        end
endmodule
```