Rubik's Cube Solver

Katharine Daly and Jack Hutchinson
6.111 Fall 2013
Final Project Report

Abstract

For our 6.111 final project, we created a system that helps a user solve a Rubik's cube from any starting condition. The starting configuration of the cube is entered onto a cube net displayed on the computer monitor by using labkit buttons to select the appropriate color for each cubelet face. Once a valid starting configuration has been provided, a 3D visualization of the Rubik's cube helps guide the user towards solving the cube. Each time a designated button on the labkit is pressed, one section of the displayed Rubik's cube rotates and the colors of the cubelet faces are updated to reflect the current state of the cube. The algorithm used to determine the appropriate sequence of moves is a well-known seven step method for solving the cube. If users accurately perform each of the rotations they see in the visualization on their physical cube, they should have a solved Rubik's cube in fewer than 200 rotations.

Table of Contents

1. Introduction

1.1. Motivation

The Rubik's Cube is a popular puzzle that many people struggle to solve. It is difficult to provide an easy-to-follow algorithm to solve the cube on a sheet of paper, so the solution manual that comes with the kit can often be tough to follow. Algorithms posted on YouTube in video format are slightly more helpful, but if the goal of the user is just to return the cube to its solved state without having to learn a generalized solving method, these videos are unnecessarily complex. The goal of our project is to provide a step-by-step animation on a computer monitor of the rotations necessary to solve a Rubik's cube from any starting configuration that the user inputs.

1.2. Overview

The user must first enter the starting configuration of the cube, and to aid this process, a cube net as shown in Figure 1 is displayed on a computer monitor. A Rubik's cube is composed of 26 smaller cubelets, and the 54 cubelet faces that are visible on the cube are displayed in this cube net. The colors of each of the center cubelet faces are fixed in the cube net, helping orient the user. It is reasonably likely that the user will make an error when inputting the cube configuration, so our system runs a check on the desired starting configuration to make sure it is solvable before allowing the user to proceed. The user cannot trigger rotations on the 3D visualization of the cube until the configuration is valid.

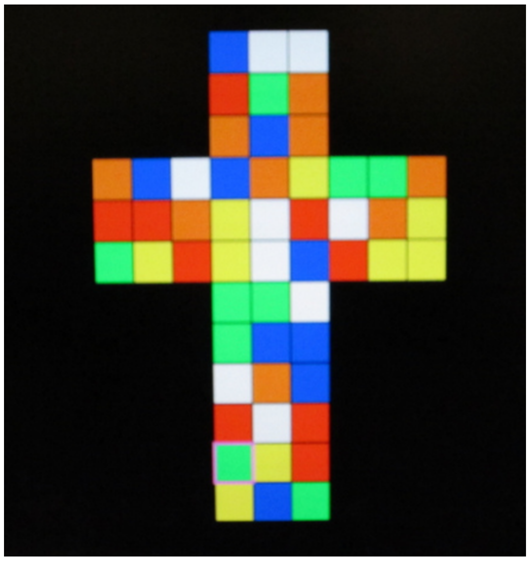


Figure 1. Cube Net. The user enters the starting configuration of the Rubik's cube on a cube net. This is a screenshot of a valid starting cube configuration.

The process of solving a cube with a valid starting configuration is communicated to the user via a 3D animation displayed on the computer monitor. A screenshot of this visualization is included in Figure 2. This animation consists of a Rubik's cube whose faces rotate to illustrate each step of the solution. Following each rotation, which the user triggers by pressing a button on the labkit, the colors of the visible cubelet faces update to reflect the current configuration of the cube. The perspective of the cube itself does not change throughout the animation, meaning that the cube as a whole will not move, but rather individual sections of the cube will move. Users should be able to return their cube to a solved state if they are able to perform each of the rotations shown in the visualization while maintaining the same cube orientation throughout.

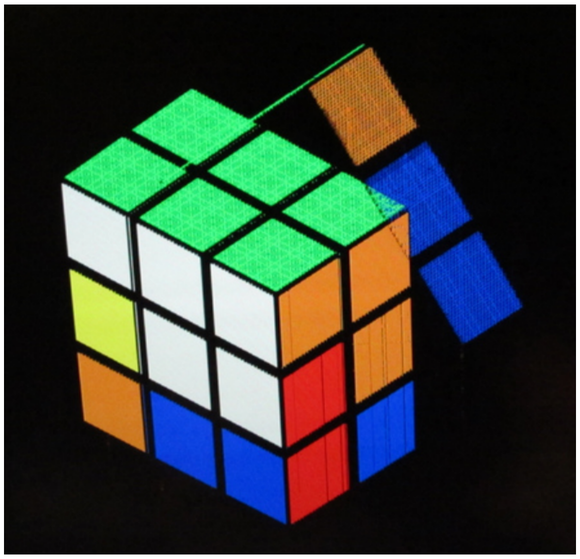


Figure 2. Animated Cube Rotation. The animated cube rotations guide the user through the solution of the cube. This is a screenshot of a rotation in progress.

Our system of course includes an implementation of an algorithm that generates the sequence of rotations that will lead to a solved cube. There exist many known algorithms for solving the Rubik's Cube, and in general there is a tradeoff between the number of potential moves that must be memorized (or stored in memory in our case) and the number of rotations that are ultimately required to solve the cube. For example, it has been shown that any Rubik's Cube configuration can be solved in twenty or fewer moves[1], but large amounts of memory would be required to store the sequence of moves for each starting configuration simply because there are 4.3252×10^{19}, or more than 40 quintillion, starting configurations[2]. The algorithm we chose requires a relatively small amount memory for move sequences, and fewer than 200 rotations are needed.

1.3. High Level Structure

Our system easily divides into three main blocks, namely the Get Initial State, Solve Cube, and Show Animation blocks, as shown in Figure 3. The Get Initial State block collects the starting configuration of the Rubik's Cube using the cube net. This information gets sent to the Solve

[1] http://www.cube20.org/
[2] http://web.mit.edu/sp.268/www/rubik.pdf

Cube block, which both checks the starting configuration to see if it is valid and determines the sequence of rotations necessary for solving the cube. The Show Animation block communicates these rotations to the user via a 3D Rubik's cube that is displayed on a computer monitor.

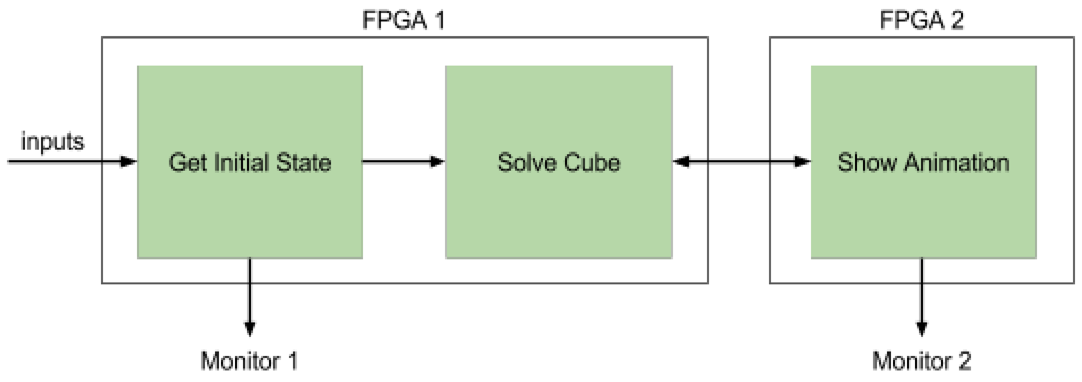


Figure 3. Block Diagram. Our implementation is divided into three basic states that gather the initial cube configuration, solve the cube, and display the animation. Two FPGAs are used and communication between the two labkits occurs over a serial connection.

We had initially planned to implement all three blocks on a single FPGA but ran into issues with limitations of the FPGA. Thus, we decided to use one FPGA to get the starting configuration and to solve the cube and a second FPGA to create the 3D animation. This enabled us to use two separate monitors, so the cube net used for input and the 3D animation are always both shown. Communication between the two FPGAs occurs over a serial connection we established between the two labkits.

To manage the communications among these three blocks, protocols are used to represent the current configuration of the Rubik's cube and to describe the rotations of various faces of the cube. These two protocols, the serial communication between the two FPGAs, and more details on the implementation of each main block are explained in the following sections.

2. Protocols

2.1. Cube Configuration Representation (Katharine)

The protocol for representing the current cube configuration encodes the color of each cubelet face and is depicted in Figure 4. It is used by the Get Initial State block to communicate the starting configuration to the Solve Cube block, and it is used by the Solve Cube block to communicate the current configuration of the cube to the Show Animation block. A cubelet is one of the 26 smaller cubes that together compose the Rubik's Cube, and since the Rubik's cube itself has 6 faces each with 9 individual cubelet faces, there are 54 total cubelet faces. The Rubik's cube has six differently colored stickers, allowing these colors to be represented with 3 bits: red (001), orange (010), yellow (011), green (100), blue (101), and white (110). The other colors used in the system are grey (000) which is used for blank cubelets, and black (111) which is used for the background and the gaps between cubelets. If a specific ordering of the cubelet faces is chosen, the entire configuration of the Rubik's Cube can be represented with a string of 6*9*3=162 bits.

The ordering that we chose relies on the orientation of each Rubik's Cube face as seen in the animation. This orientation is constant, for the cube as a whole never rotates during the animation, but rather movements are restricted to the rotation of certain small sections of the cube. The six orientations of the faces are up, down, front, back, right, and left. It is important to note that the colors of center cubelets cannot be used to define the orientation of the cube, as rotations of the middle sections of the cube move these center cubelets from one orientation to another.

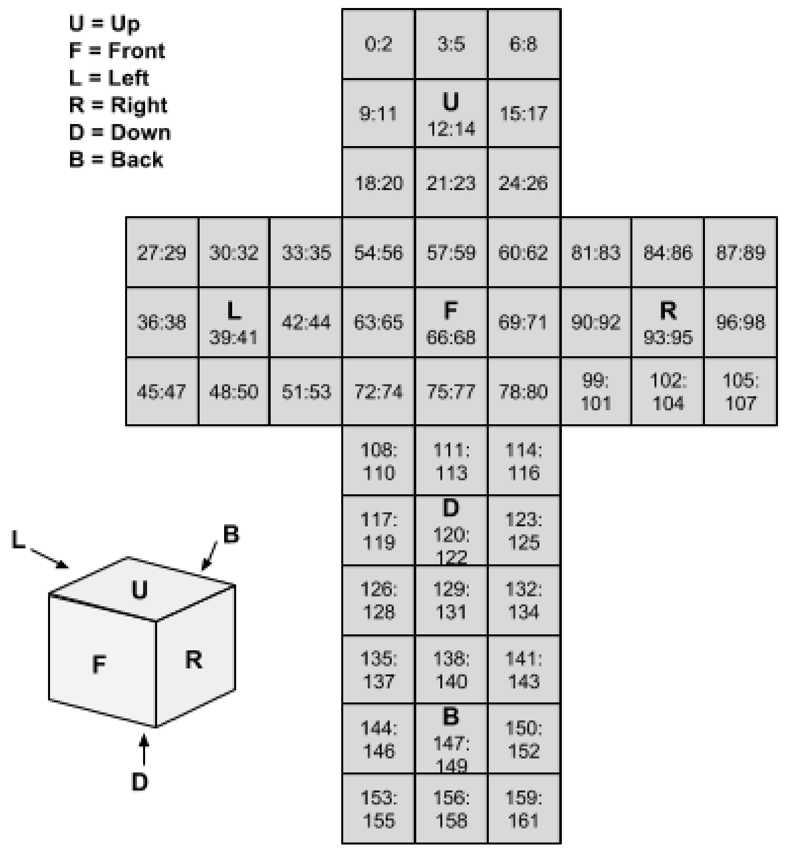
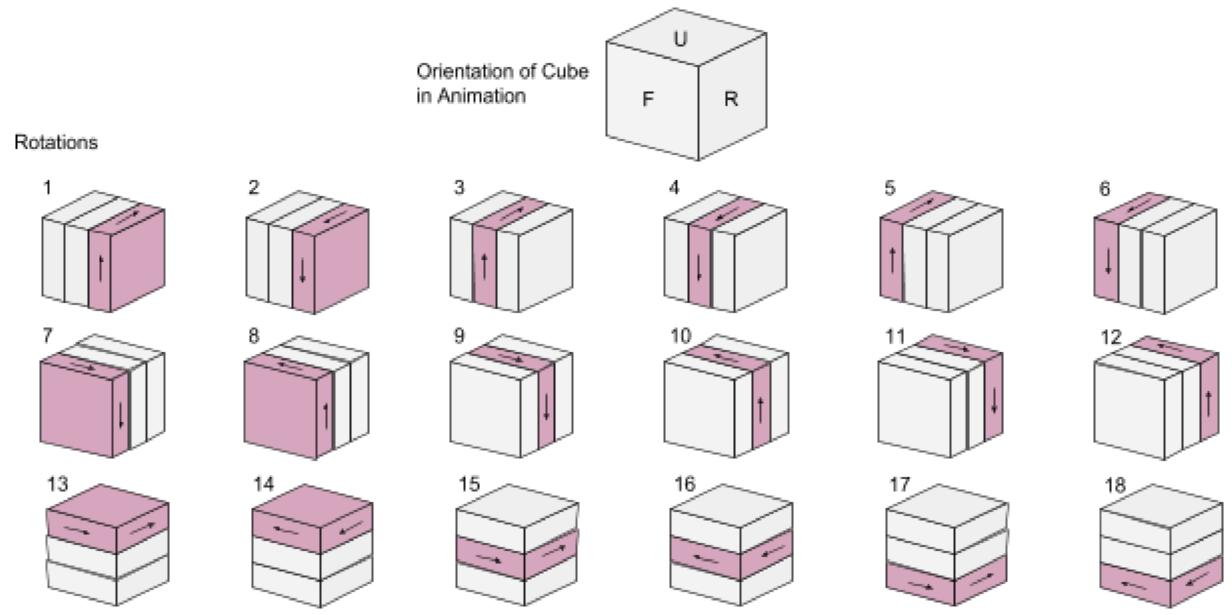


Figure 4. Rubik's Cube Configuration Protocol. A 162 bit string is used to represent the configuration of the cube, where each 3 bit instance corresponding to a specific cubelet face encodes one of the six possible sticker colors. The directions of the faces are used to orient the configuration.

When the starting configuration is provided by the user, the center cubelet faces in the cube net displayed on the computer monitor have a fixed color. This places these center cubelet faces in a pre-determined starting orientation. For example, the green center cubelet is required to face upwards and the white center cubelet is required to be in front in the starting configuration. This was done to help initially orient the user, and it also made implementing the Solve Cube block easier.

2.2. Rotation Representation (Katharine)

The second protocol assigns numerical values to the possible rotations that can occur, and the Solve Cube and Show Animation blocks rely on this protocol to communicate effectively. There are 18 possible rotations that are depicted in Figure 5 along with their assigned number, which can be encoded with 5 bits, and their assigned variable name. An example of an assigned variable name is MFR (an acronym for Move Front Face Right), which corresponds to rotating the entire front face clockwise.



1: MRU: Move Right Face Up
2: MRD: Move Right Face Down
3: MFUBD: Move Front Face Up and Back Face Down
4: MFDBU: Move Front Face Down and Back Face Up
5: MLU: Move Left Face Up
6: MLD: Move Left Face Down
7: MFR: Move Front Face Right
8: MFL: Move Front Face Left
9: MURDL: Move Up Face Right and Down Face Left
10: MULDR: Move Up Face Left and Down Face Right
11: MBR: Move Back Face Right
12: MBL: Move Back Face Left
13: MUR: Move Up Face Right
14: MUL: Move Up Face Left
15: MFRBL: Move Front Face Right and Back Face Left
16: MFLBR: Move Front Face Left and Back Face Right
17: MDR: Move Down Face Right
18: MDL: Move Down Face Left

Figure 5. Rubik's Cube Rotation Protocol. There are 18 distinct rotations that can be encoded as 5 bit numbers in the communication from the Solve Cube block to Show Animation block.

2.3. FPGA-to-FPGA Communication (Jack)

Having tested each of the main blocks individually we tried to integrate them together and compile the verilog. We were able to successfully integrate the Get Initial State block and the Solve Cube block together. However, when we tried to combine all three modules, due to the computationally intensive nature of the Show Animation block, there were issues with routing when the compiler tried to implement the design on the FPGA. Therefore, we decided to use one

FPGA for the first two blocks and a second FPGA for the Show Animation block. However, before each rotation, the Solve Cube block needed to send the 81 bit representation of the visible faces of the next cube configuration and the 5 bit representation of the next rotation to be performed to the Show Animation block. Upon completion, the Show Animation block had to send a ready signal to indicate to the Solve Cube block to work out the next rotation and cube configuration. Therefore we needed to come up with a protocol for communication between the two FPGAs.

We considered two possible methods to implement this communication. The first was to send the data serially, which is slow but required only four wires between the two FPGAs. The second method was to send the data via around 87 wires (one wire for each bit of data sent) which would be much quicker but more error prone and harder to debug due the the large number of wires running between the two FPGAs. Since speed was not a concern because the blocks only communicate once every two seconds, we decided to send the data serially from one FPGA to the other.

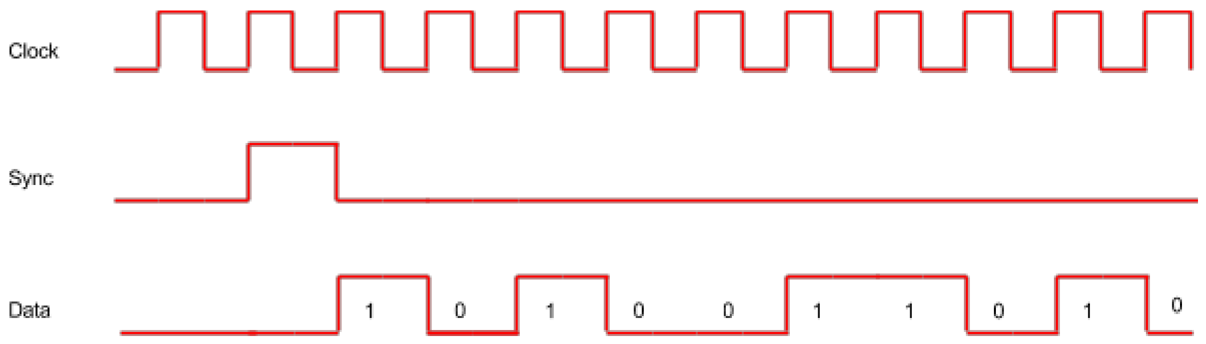


Figure 6 - **Serial Communication Signals**. The signals shown in the diagram correspond to the signal transmitted in 3 of the wires linking the two FPGAs.

Figure 6 shows the signals that are sent in three of the wires. The clock of FPGA 1 is sent to FPGA 2 via the first wire. This clock is much slower than the internal clock (about 100 kHz) to ensure that the data is reliably transmitted by making clock skew due to the length of the wires negligible. Sending the clock from FPGA 1 to FPGA 2 ensures that the internal clocks of the FPGAs don’t have to be in sync for the communication to be reliable. When FPGA 1 is ready to send data it sends a sync pulse through the second wire and then sends that data serially via the third wire. Both the sync pulse and data pulse are synced with the positive edge of the clock and each pulse has a duration of one full clock period. When FPGA 2 receives the sync pulse it begins to shift the received data into a shift register until all of the data has been received. In order to ensure that the data is received correctly, each new bit of data is shifted in on the negative edge of the clock. This ensures that the data is received extremely reliably, regardless of if there is clock skew.

The fourth and final wire is used to transmit a ready signal from FPGA 2 back to FPGA 1. FPGA 1 treats this signal input like a button and so debounces it to prevent errors. Other than that there are no special protocols for sending and receiving this signal.

3. Get Initial State Block (Jack)

The Get Initial State block's purpose is to allow the user to quickly and easily enter their specific unsolved cube configuration into the system so that the Solve Cube and Animation Blocks could then walk them through the solution. This is done by displaying two dimensional representation of the cube on the screen (see Figure 7) which the user can modify as appropriate. The user presses a button to indicate the configuration is finished being entered at which point the system checks to ensure the configuration is valid. Control is then passed to the Solve Cube and Show Animation Blocks.

The Get Initial State block consists of one main module containing the controlling FSM which calls the blob module used in lab 3 to display each of the cubelet faces, and a slight variation of the blob module to display the pink border that indicates the currently selected cubelet face. Each cubelet face corresponds to a state in the FSM and the current state corresponds to the currently selected state.

Initially all of the faces are set to be grey, apart from the centre cubelets on each face which are predesignated, as shown in Figure 7, and the user cannot modify these cubelets. The top left hand green face square is initially selected. For each selected cubelet the user has five options which control what the next FSM state will be (i.e which cubelet to select next):

- Left
- Down
- Right
- Up
- Cycle colour

As one might expect, pressing the left button selects the cubelet to the left of the currently selected one. The same occurs for down, right and up. The selector wraps around too so for example if a right edge cubelet is selected and the user presses the right button the selected cube will now be the left edge cube on the same vertical line. Also, the selector jumps over the centre cubelets as the user cannot modify these. If the cycle color button is pressed the colour of the selected cubelet is changed to the next colour in the cycle (e.g. red then orange then yellow etc.). The FSM remains in the current state but the appropriate bits in the cube configuration register are updated. Once the colour has

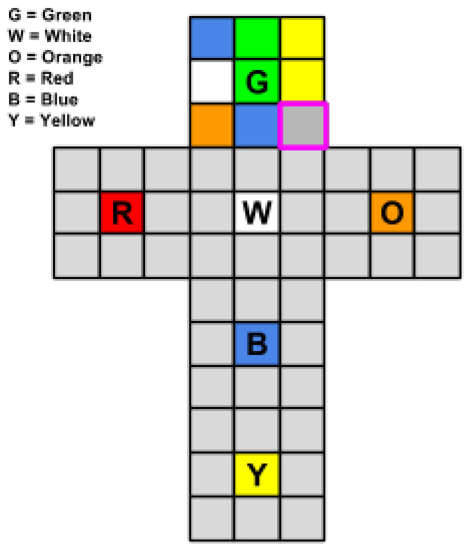


Figure 7 - **Input Initial Configuration Screen.** User can navigate between the cubelet faces using the arrow buttons on the labkit (selected face highlighted by pink box outline) and input the desired colour by cycling through the 6 options using the appropriate button.

been changed from grey it cannot be changed back to grey (i.e it cycles straight from white to red).

Once the user is satisfied with the configuration of the cube, they can press a button to check that the configuration they have entered is in fact valid. At this point the Solve Cube block, using the configuration stored in the current configuration register, attempts to solve the cube. If it is successful, the configuration must be valid and it notifies the user via the hex display. If it is unable to solve the configuration then the user must have entered the configuration incorrectly and so must edit the configuration and try the check again. When the user is ready they can press another button to start the process by which the solution is animated step by step.

4. Solve Cube Block (Katharine)

The Solve Cube block determines the sequence of rotations necessary for solving the provided Rubik's Cube starting configuration. The logic behind generating this rotation sequence parallels that of a well-known 7-step process for solving the cube layer by layer that is described in Table 1 below. The rotation sequences for the first two steps were worked out by hand, and the sequences for the latter steps were constructed based on a methodology presented in a wikiHow article on solving a Rubik's Cube[3].

To aid in the understanding of each of the steps, it is helpful to define several different types of cubelets. Edge cubelets are cubelets with only two colored faces, corner cubelets are cubelets with three colored faces, and a cubelet is in the correct position only if in-place changes of orientation can achieve the ultimate solved configuration. Each step in the process represents a sequence of rotations with a specific desired outcome.

Step	Description	Number of Repetitions
1	Correct the position and orientation of a single edge cubelet on the top layer	4 (There are four edge cubelets on the top layer)
2	Correct the position and orientation of a single corner cubelet on the top layer	4 (There are four corner cubelets on the top layer)
3	Correct the position and orientation of an edge cubelet on the middle layer	4 (There are four edge cubelets on the middle layer)
4	Correct the position of all corner cubelets on the bottom layer	1 (All four corner cubelets on the bottom layer are dealt with simultaneously)
5	Correct the orientation of all corner cubelets on the bottom layer	1 (All four corner cubelets on the bottom layer are dealt with simultaneously)

[3] http://www.wikihow.com/Solve-a-Rubik's-Cube-(Easy-Move-Notation)

6	Correct the position of all edge cubelets on the bottom layer	1 (All four edge cubelets on the bottom layer are dealt with simultaneously)
7	Correct the orientation of all edge cubelets on the bottom layer	1 (All four edge cubelets on the bottom layer are dealt with simultaneously)

Table 1. Seven Step Solution. The algorithm we implemented follows a seven step solution process. The steps correct the position and/or orientation of specific cubelets and are to be performed in order, with some steps being repeated four times.

The Solve Cube block itself is composed of a solution_state_machine module, a find_cubelet module, a rotate module, a send_serial module, and seven additional modules corresponding to the seven steps of the solution process from Table 1. These modules and their communication paths are shown in Figure 8. The solution_state_machine module tracks both the progression towards the solved Rubik's Cube configuration and whether or not the starting configuration provided by the user is valid. The find_cubelet module locates a color-specified cubelet, and the rotate module modifies the representation of the current configuration to mimic a rotation. The seven modules that correspond to the seven solution steps produce strings of rotation sequences that can be used to complete a given step. Finally, the send_serial module relays the rotation type and current cube configuration information to the Show Animation block. These modules will be discussed in more detail in the following subsections.

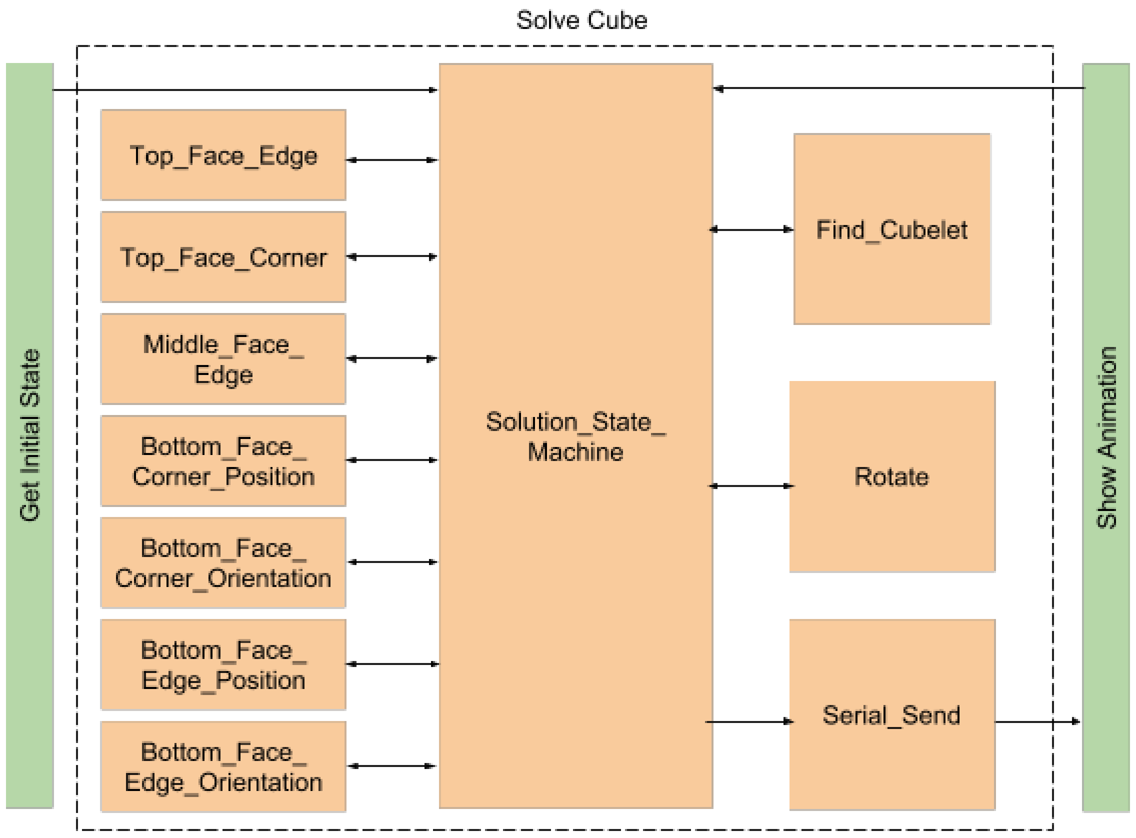


Figure 8. Modules within Solve Cube block. The Solve Cube block contains eleven modules; seven of these select the appropriate move sequence for a specific step in the solution, a state machine module tracks the progression of the solution and whether or not the starting configuration has been verified, a find cubelet module locates a color-specified cubelet, a rotate module alters the cube configuration representation, and a serial_send module sends rotation and configuration information to the Show Animation block.

4.1. Solution_State_Machine Module (Katharine)

The solution_state_machine module guides the solving process of the cube and is the central component of the Solve Cube block. It progresses linearly, calling the seven modules corresponding to the seven steps to the solution listed in Table 1 the correct number of times and in the correct order. There are seventeen states in total, which are displayed in the representation of the state machine in Figure 9. The work done at each state typically involves calling the find_cubelet module to locate a cubelet or multiple cubelets, using one of the seven step-specific modules to determine the appropriate sequence of rotations, and then calling on the rotation module once for each these rotations. A transition from one state to the next occurs whenever all of the rotations for that step have been completed, and there is only one transition possible per state. To start the solving process, a user presses a designated labkit button on FPGA 1, and to trigger a new rotation, a user presses a different button on FPGA 2. Pressing the button corresponding to a new rotation causes a signal to be sent over a wire from the labkit running the Show Animation block to the labkit running the rest of the modules. We found it necessary to put a debouncer on this wire for the signal to be accurately interpreted.

One design decision we made to simplify the coding of the algorithm was to always treat the face with the green center as the top layer and to correct cubelets in a specific order, as shown in the state machine. This rigid order of solving the cube is most straightforward in terms of implementation, but it is not necessarily ideal in terms of minimizing the number of rotations needed to solve the cube. For example, suppose a user takes a solved Rubik's Cube and rotates the up face (the green face) to the right once. Clearly only one move is needed to solve the Rubik's Cube, but by following the fixed progression described in the state machine, it can be seen that the system will not detect the easy solution, leading to many more rotations.

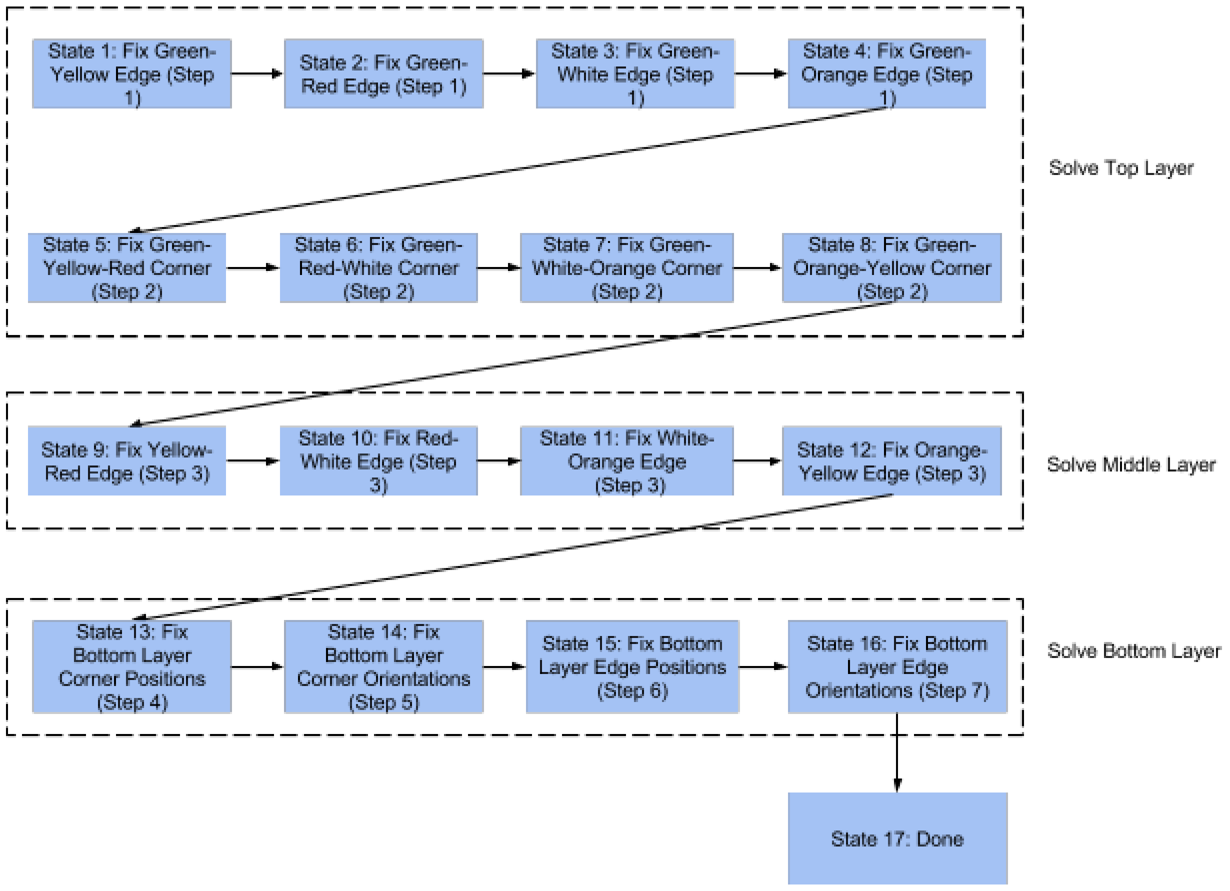

State 1: Fix Green-Yellow Edge (Step 1)
State 2: Fix Green-Red Edge (Step 1)
State 3: Fix Green-White Edge (Step 1)
State 4: Fix Green-Orange Edge (Step 1)
Solve Top Layer
State 5: Fix Green-Yellow-Red Corner (Step 2)
State 6: Fix Green-Red-White Corner (Step 2)
State 7: Fix Green-White-Orange Corner (Step 2)
State 8: Fix Green-Orange-Yellow Corner (Step 2)
State 9: Fix Yellow-Red Edge (Step 3)
State 10: Fix Red-White Edge (Step 3)
State 11: Fix White-Orange Edge (Step 3)
State 12: Fix Orange-Yellow Edge (Step 3)
Solve Middle Layer
State 13: Fix Bottom Layer Corner Positions (Step 4)
State 14: Fix Bottom Layer Corner Orientations (Step 5)
State 15: Fix Bottom Layer Edge Positions (Step 6)
State 16: Fix Bottom Layer Edge Orientations (Step 7)
Solve Bottom Layer
State 17: Done

Figure 9. Solution State Machine. The process of solving the Rubik's Cube can be broken down into 17 states. The figure indicates which cubelets are fixed in each state as well as which of the seven solution steps each state corresponds to. There is only one transition out of each state, and this transition occurs when all the rotations necessary for fixing the cubelet(s) have been completed.

In addition to generating the sequences of rotations that are ultimately reproduced on the 3D animation of the cube, the solution_state_machine module is responsible for checking whether a cube configuration is valid. This can be done simply by seeing if, starting from the first state in the state machine, the done state can be reached without error. One type of error occurs when the find_cubelet module searches for a specific cubelet within the cube configuration but cannot find it. For example, the state machine may be in state 11, trying to fix the edge cubelet with white and orange faces, but the starting cube configuration entered by the user may not contain such a cubelet. The second type of error occurs when one of the seven step-specific modules is not able to find a rotation sequence that will transform the current position and/or orientation of specific cubelets into the desired position and/or orientation. An instance in which this error might occur is with a Rubik's cube that is completely solved except for one corner, as it is not possible for only one corner to have the wrong orientation while the other three corners are correct.

Since the validation check should occur quickly, rotations during validation are triggered in the module itself, not by button presses on the second labkit. This calls for a relatively simple secondary state machine composed of three main states: not verified, verifying, and verified. A system that has just been reset is in the not verified state, and it enters the verifying state once a user presses a designated button on the first labkit signalling they would like to check their inputted configuration. Once in this verifying state, pulses are created at a certain frequency to trigger rotations that will allow for progression through the primary state machine that is responsible for solving the cube. These pulses were separated 256 clock cycles apart to allow ample time to decide which rotation should be next and update the cube configuration to reflect this rotation. If the done state in the solution state machine is reached without any of the errors described above, then the secondary state machine transitions to the verified state. Once in this verified state, the user is able to reload the starting configuration onto the cube and then use a button on the second labkit to see individual rotations in the 3D cube animation. If an error is detected while checking the cube configuration, the secondary state machine transitions back to the not verified state and the user is able to modify the starting cube configuration in the cube net.

4.2. Find_Cubelet Module (Katharine)

The find_cubelet module finds the location of either an edge cubelet or corner cubelet that is specified by the color of its faces within the 162 bit representation of the cube configuration. The location that is returned is an encoding of the orientations (up, down, right, etc) of each of the color-specified cubelet faces. Since this module is used by the solution_state_machine module, which often works with a cube orientation where the green center cubelet is not necessarily on the up face (as opposed to the displayed animation where the green center cubelet is always on the up face), the find_cubelet module returns the encoded orientations based a specified orientation of the cube. This cube orientation is passed into the module with a bit string that encodes the color of each of the cube faces in a specific order.

4.3. Rotate Module (Katharine)

The rotate module alters the cube configuration representation for all types of rotations and sends rotation protocol commands to the Show Animation block. As explained in the section on the rotation protocol, there are eighteen different types of rotations that are specified using 5 bits. A caveat is that when the solution_state_machine desires a specific rotation, this rotation may be defined relative to a cube orientation where the green center cube is not necessarily on the up face (as opposed to the animated display). Thus, the rotate module must be able to determine the correct 5-bit rotation to send to the Show Animation block, which will be assuming that the rotation is specified according to a cube orientation with a green center cube on the up face, so that the desired set of cubelets is ultimately moved.

4.4. Algorithm Step Modules (Katharine)

As listed earlier in Table 1, there are seven steps in the solution, each of which has its own module that determines what sequence of rotations should be performed to complete a certain state in the state machine. The construction of these modules is quite uniform, and they all take as input the location of specific cubelets. Some step modules require knowledge of the position of one cubelet while others require knowledge of four cubelet positions. The main part of each of these modules is a case statement that matches the cubelet position(s) to a sequence of rotations.

In some cases where many of the rotation sequences for a particular step differed only slightly, it was helpful to define a subsequence of rotations. Also, to limit the number of values enumerated under the case statements, it was possible in some steps to disregard some of the cubelet position information. For example, in state 13, which fixes the bottom layer corner positions, the specific orientation of each face on the blue-yellow-red cubelet, for example, is not needed. Rather it is sufficient to know if this corner cubelet is on the top left, top right, bottom left, or bottom right when the bottom face of the cube is looked at directly.

These rotation sequences produced by each of the step modules are no more than 40 rotations long, and each rotation can be encoded in 5 bits as specified by the rotation protocol. The longest rotation sequence at 38 rotations is in the very last step module that fixes the orientations of the edges in the bottom layer. In general, the lengths of the rotation sequences across the other step modules are much shorter and are typically fewer than 15 rotations long.

4.5. Serial_Send Module (Katharine)

The serial_send module establishes the one-way serial communication from the Solve Cube block to the Show Animation block. The information sent includes both the 5 bit rotation type and the current 162 bit cube configuration, and the module follows the protocol described earlier in the section on FPGA-to-FPGA communication.

4.6. Testing (Katharine)

As there are so many possible starting configurations of the Rubik's cube, it is not possible to do an exhaustive testing of the Solve Cube block. However, testing all configurations is not necessary as the solution is divided into seven types of rotation sequences corresponding to each step in the solution. The first step, which moves an edge cubelet into its correct position on the top face of the cube, selects one rotation sequence from 24 possible rotation sequences. Thus, to adequately test the first step, all 24 possible rotation sequences would ideally be tested.

Similarly, the other six step modules select one rotation sequence from a relatively small number of pre-entered rotation sequences with the help of a case statement that looks at the position of the cubelet(s) being corrected. In order to test the majority of the possible rotation sequences from each step and catch any errors caused by typing in the sequence incorrectly, thirty different starting configurations were tested. A handful of errors were caught in the first few tests, but the last twenty or so configurations were solved successfully, indicating a high probability that the Solve Cube block could solve any valid starting configuration.

To help debug the interaction between the Get Input State block and Solve Cube block and that between the Solve Cube block and the Show Animation block, several values were displayed on the hex display of the first labkit in decimal format. The first value indicates the state of the secondary state machine, that is whether or not the starting configuration has been deemed valid or not. The second value displays the state of the solution state machine (1 to 17), and the third value displays the rotation that should next occur (1 to 18).

5. Show Animation Block (Jack)

The Show Animation block does exactly as one might expect: it displays the cube on the screen and directs the users, by means of animated cube rotations, how to solve the cube. The block renders the three dimensional representation of the cube configuration sent to it by the Solve Cube block in terms of a two dimensional pixel map and animates the solution step rotations when required. This information must then be converted appropriately and sent to the display. The Animation Block performs these functions using four main submodules as shown below in Figure 10.

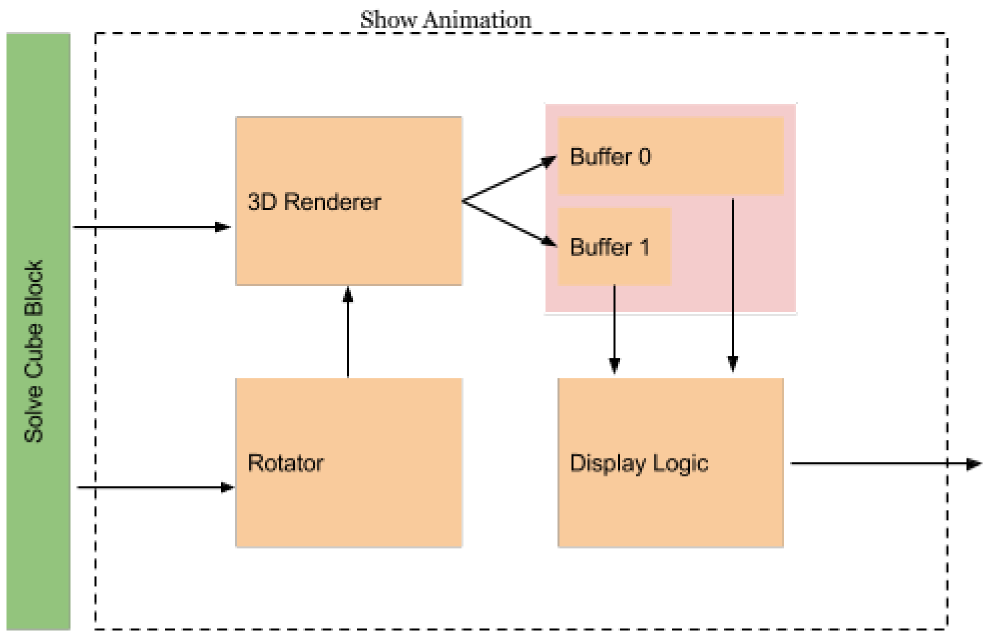


Figure 10. Show Animation Block. This block contains four submodules. The rotator, 3D renderer and Display logic perform operations to generate the display while the buffers provide

memory to store the next frame to be displayed. The Show Animation block takes in the cube configuration and the rotation type from the Solve Cube block and renders an image that is then sent to the ADC7125 chip on the FPGA to be sent to the monitor.

5.1. 3D Renderer and Buffer (Jack)

These submodules together take in the cube configuration, render a 3D representation of it and store it so that it can be sent to the display in the next frame. The cube is represented in 3D space by means of x, y and z coordinates. Each cubelet face is represented by a 100x100 square of coloured points, generated by a version of the blob module from lab 3 modified to operate based on three coordinate inputs instead of two. These 3D coordinates are then mapped into 2D by simple matrix multiplication. When faces are being rotated, the 3D renderer pre multiplies the coordinates by the relevant rotation matrix selected for each face by the rotation module and made up of the cos and sin values sent by the rotation module before the 3D to 2D mapping takes place. The following are the rotation matrices that are used:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

All of this is actually performed for each cubelet face by a submodule and then the 3D renderer module pieces all of this information together and sends it to the RAM. Rendering the 3D cube to be displayed on the 2D screen will not happen quickly enough or in the required order to be displayed instantly on screen, so the generated pixel map is stored in a buffer so that it can be displayed in the next frame. Two buffers are required, one to store the output of the 3D renderer and one from which the display logic reads. The role of each buffer switches every frame. Since multiple 3D points can be mapped to a single 2D pixel, the module ensures that nearer points have precedence over more distant points. This is done by saving a five bit representation of the pixels depth in the buffer along with the the pixel colour. Also, before the pixel data is overwritten for a particular location, the depth is first compared to the depth of the pixel already in the buffer and the information will only be replaced if the new pixel is nearer. Each buffer is approximately 312000 pixels in size (the cube is about 520 x 600 pixels on screen) and takes up approximately 5 Mbits.

5.2. Rotator (Jack)

The rotator module is a finite state machine that performs the rotation animations one frame at a time. The module rotates segments of the cube when signalled to do so by the Solve Cube block (as shown in Figure 11). Depending on which of the 18 rotations is to be carried out, the module selects the relevant faces to be rotated and also the axis about which they should be rotated and sends this information to the 3D render module. These rotations are then carried out in 2 seconds and so require about 30 frames to create the optical illusion of a moving image. For each frame, the module sends the correct sine and cosine factors to the 3D renderer submodule. Since there are only 30 different possible angles (30 frames) that the rotated segment can be at, the rotation matrix values can be stored in memory and need not be calculated by the module. Figure 11 shows four frames of an example rotation animation. Once a rotation has been completed the module sends a signal to the Solve Cube block that it is ready to animate the next step in the solution.

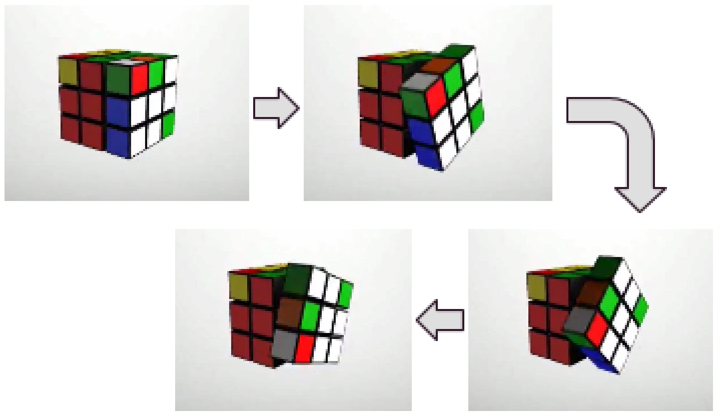


Figure 11 - **Animation of Rotations**. Four frames of an example rotation animation. The face is rotated by changing the matrix multiplication factors sent to the 3D renderer.

5.3. Display Logic (Jack)

The display logic is a very simple module that takes the values from the buffer and sends the correct pixel to the ADC7125 based on the current value of hcount and vcount. This module is very similar to the one implemented in lab 3.

5.4. Problems (Jack)

A number of problems were encountered when implementing this block, some of which were overcome and some of which required more time and FPGA experience to fix. These included ZBT memory timing, displaying only nearer pixels, calculation timing issues and multiplication rounding errors.

The ZBT memory requires special timing allowances which hadn't been allowed for when we initially designed the system. Since we needed to compare every new byte written to the memory to the old byte before writing any data to RAM, allowing for this significantly slowed down the speed at which a buffer could be generated. This meant we couldn't generate a frame buffer every frame. We overcame this problem simply by switching between the buffers once a frame had been completely generated, regardless of when this occurred, and continued to read from the other buffer for as long as necessary. The only difference this made to the final result was that it slowed down the maximum frame rate of the animations but since the screen refresh rate is 60 Hz which is far faster than the eye can detect, even if a new buffer was only generated once every 3 or 4 refreshes a fluid animation was possible.

Displaying only the closest object on the screen proved very problematic. This was because only 5 bits of depth information was stored, even though the actual depth required more like 9 bits. This meant that there were very large rounding errors in the depth and because for the most part two objects in the same 2D location were not very far apart in terms of depth, the object was displayed was usually the last one determined. This could be improved by increasing the buffer size and storing more depth bits.

The fact that a 65 MHz clock was required to generate the 1024 by 760 display meant that there was only 15ns between each rising clock edge. Since a multiplication took on the order of 8 ns, there wasn't a lot of time for routing delays or any other logic to be performed in the same clock cycle. Therefore, every now and then a calculation would be too slow and glitches would appear on the screen. Reducing the screen resolution and clock speed would have solved this problem.

6. Further Work

6.1. Improvements to Get Initial State Block (Jack)

Given more time or another team member, we would have liked to implement a system that could use a camera feed to automatically populate the initial cube configuration string. Therefore, if conducting further work on the project we would try to implement this. To do this we would build a small hexagonal mount shown in Figure 12 on which the cube would be placed for reading, with the camera underneath the mount. This would allow the camera to read three sides at once and would mean that the camera could read the colours from the same point every time without having to track the cubelets. The module would read the cube when prompted to do so by a user input. It would then read in the other three sides. The module would have to take in the raw camera data and first decode the NTSC signal input. It would then have to convert from RGB to HSV to make colour identification easier. At this point it would identify the the colour of each individual cubelet. The module would then ensure that a valid cube configuration has been entered, display the configuration on screen and ask the user to accept the configuration before signalling to the Solve Cube block that it is ready to begin solving.

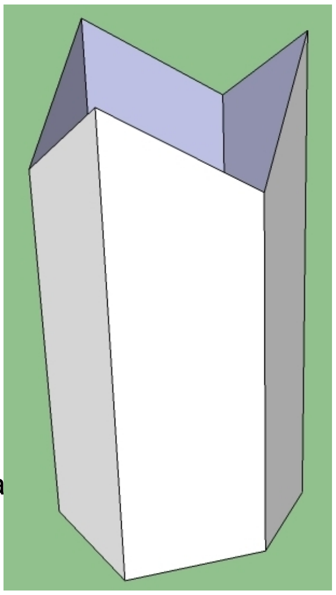


Figure 12. Proposed Cube Mount. A camera underneath the mount would read three sides of the initial cube configuration.

6.2. Improvements to Solve Cube Block (Katharine)

As discussed previously, there are methods of solving the Rubik's Cube that require far fewer rotations than our implementation does. Upgrading to one of these algorithms would involve enumerating more cases in various step modules. These cases would likely correspond to the various permutations of more than four cubelets. A more mundane approach to reducing the number of rotations is to eliminate the instances of redundant rotations. For example, on occasion, the Rubik's cube will rotate in a certain direction only to have the next rotation cancel out this change. These instances of subsequent rotations that cancel each other out are due to the adding of prefixes and suffixes to common subsequences in the step modules or are the result of complementary beginnings and endings of rotation sequences from two different step modules. Generating the entire rotation sequence at one time, rather than generating the next sequence once the current sequence has completed, would allow for removal of all redundant rotations with a single iteration through the entire sequence.

6.3. Improvements to Show Animation Block (Jack)

Going forward, the first improvements that could be made with the Show Animation block would be to eliminate all of the problems that currently exist that were outlined above. Assuming that this could be done successfully, I would also like to make it possible for the user to change the viewing angle of the cube so that if users get stuck they can more closely examine the configurations, rather than just being able to see three sides. Allowing the user to vary the speed of rotations and allowing them to play a specific rotation again would also be useful features which would greatly improve the system's user friendliness.

7. Conclusion (Katharine)

We were able to create a basic system that takes in a starting Rubik's cube configuration, checks it to ensure that it is valid, and then animates rotations on a 3D visualization of the cube to guide the user towards the solution. Our system runs on two FPGAs and uses two monitors to display the cube net where the configuration is initially entered and the 3D cube visualization which shows the rotations. Future work on the system would likely consist of improving the quality of user interaction by having a camera capture the initial state, reducing the number of rotations needed to solve the cube, and making the visualization cleaner.

Appendix A: Verilog

FPGA 1:

Get Initial State Block (Jack):

- Input_State.v

Solve Cube Block (Katharine):

- Solution_State_Machine.v
- Find_Cubelet.v
- Rotate.v
- Serial_Send.v
- Top_Face_Edge.v
- Top_Face_Corner.v
- Middle_Edge.v
- Bottom_Corner_Position.v
- Bottom_Corner_Orientation.v
- Bottom_Edge_Position.v
- Bottom_Edge_Orientation.v
- BCD.v

FPGA 2:

Show Animation Block (Jack):

- zbt_6111_sample.v
- zbt_6111.v
- threeDrenderer.v
- face.v
- rotator.v
- receiver.v

Common to Both FPGAs:

- debounce.v
- display_16hex.v
- labkit.ucf

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
// Modifications made by Katharine and Jack
// Description: Main module for running the input state and solve cube blocks
// on the first FPGA
//
///////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2012-Sep-15: Converted to 24bit RGB
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
```

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

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

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

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

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

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

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

           clock_feedback_out, clock_feedback_in,

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

           rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

           mouse_clock, mouse_data, keyboard_clock, keyboard_data,

           clock_27mhz, clock1, clock2,

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

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

           switch,
           led,

           user1, user2, user3, user4,

           daughtercard,

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

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

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

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

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

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

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

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

   input  clock_feedback_in;
   output clock_feedback_out;

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

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

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

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

   inout [43:0] daughtercard;

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

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

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

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

   // Video Output
   assign tv_out_ycrcb = 10'h0;
   assign tv_out_reset_b = 1'b0;
   assign tv_out_clock = 1'b0;
   assign tv_out_i2c_clock = 1'b0;
   assign tv_out_i2c_data = 1'b0;
   assign tv_out_pal_ntsc = 1'b0;
   assign tv_out_hsync_b = 1'b1;
   assign tv_out_vsync_b = 1'b1;
   assign tv_out_blank_b = 1'b1;
   assign tv_out_subcar_reset = 1'b0;
   // Video Input
   assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b0;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

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

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

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

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

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

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

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

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

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

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

   // button 3 button is user reset
   wire reset,user_reset;
   debounce db1(.reset(power_on_reset),.clock(clock_65mhz),.noisy(~button3),.clean(user_reset));
   assign reset = user_reset | power_on_reset;

   // up, down, left, right, and button 0 used for input state block
   wire up,down,left,right,enter,colour_change;
   debounce db2(.reset(reset),.clock(clock_65mhz),.noisy(~button_up),.clean(up));
   debounce db3(.reset(reset),.clock(clock_65mhz),.noisy(~button_down),.clean(down));
   debounce db4(.reset(reset),.clock(clock_65mhz),.noisy(~button_left),.clean(left));
   debounce db5(.reset(reset),.clock(clock_65mhz),.noisy(~button_right),.clean(right));
   debounce db6(.reset(reset),.clock(clock_65mhz),.noisy(~button_enter),.clean(enter));
   debounce db7(.reset(reset),.clock(clock_65mhz),.noisy(~button0),.clean(colour_change));

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

   wire [161:0] cube_config_packed;
   assign analyzer1_data = cube_config_packed [145:130];
   assign analyzer3_data = cube_config_packed [161:146];

   // feed XVGA signals to input state block
   wire [23:0] pixel;
   wire phsync,pvsync,pblank;
   Input_State INSTATE(.vclock(clock_65mhz),.reset(reset),
        .up(up),.down(down),.left(left),.right(right),.enter(enter),.colour_change(colour_change
        ),
        .hcount(hcount),.vcount(vcount), .hsync(hsync),.vsync(vsync),.blank(blank),
        .cube_config_packed(cube_config_packed),.phsync(phsync),.pvsync(pvsync),
        .pblank(pblank),.pixel(pixel));

   // use button 1 to allow the user to check the configuration
   // debounce the next rotation signal sent from the second labkit
   wire start, next;
   debounce db8(.reset(reset), .clock(clock_27mhz), .noisy(~button1), .clean(start));
   debounce db9(.reset(reset), .clock(clock_27mhz), .noisy(user1[3]), .clean(next));

   // create the solve cube block
   wire [0:161] cube_config;
   wire [4:0] animation_type;
   wire [4:0] state;
   wire [2:0] run_state;
   wire clock_pulse, sync_pulse, serial_data;
   Solution_State_Machine algorithm (.clk(clock_27mhz), .reset(reset), .start(start),
        .start_cube_config(cube_config_packed), .next(next), .cube_config(cube_config),
        .animation_type(animation_type), .state(state), .run_state(run_state),
        .clock_pulse(clock_pulse), .sync_pulse(sync_pulse), .serial_data(serial_data));
   // display solution state information on the hex display
   wire [63:0] data;
   display_16hex hexdisplay (.reset(reset), .clock_27mhz(clock_27mhz), .data(data),
        .disp_blank(disp_blank), .disp_clock(disp_clock), .disp_rs(disp_rs),
        .disp_ce_b(disp_ce_b), .disp_reset_b(disp_reset_b), .disp_data_out(disp_data_out));

   // display the rotation type and solution state in decimal format
   wire [3:0] animation_type_tens, animation_type_ones, state_tens, state_ones;
   BCD animation_type_bcd (.binary(animation_type), .tens(animation_type_tens), .ones(
   animation_type_ones));
   BCD state_bcd (.binary(state), .tens(state_tens), .ones(state_ones));

   // show which absolute rotation is next
   assign data[3:0] = animation_type_ones;
   assign data[7:4] = animation_type_tens;
   // show the current state (1-17) of the solution
   assign data[19:16] = state_ones;
   assign data[23:20] = state_tens;
   // show whether the starting configuration has been verified
   assign data[34:32] = run_state;


   assign user1[2] = serial_data;       // send rotation and cube configuration over serial to
   the second labkit
   assign user1[1] = sync_pulse;        // send a sync pulse
   assign user1[0] = clock_pulse;       // send a relatively slow clock square wave

   reg [23:0] rgb;
   reg b,hs,vs;
   always @(posedge clock_65mhz) begin
        hs <= phsync;
        vs <= pvsync;
        b <= pblank;
        rgb <= pixel;
   end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_65mhz.
   assign vga_out_red = rgb[23:16];
   assign vga_out_green = rgb[15:8];
   assign vga_out_blue = rgb[7:0];
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_blank_b = ~b;
   assign vga_out_pixel_clock = ~clock_65mhz;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   assign led = ~{3'b000,up,down,reset,switch[1:0]};

endmodule

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

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

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

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

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

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

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

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

////////////////////////////////////////////////////////////////////////////////
// Author: Jack
////////////////////////////////////////////////////////////////////////////////

module Input_State (
            input vclock,           // 65MHz clock
            input reset,            // 1 to initialize module
            input up,               // 1 to select above cubelet
            input down,             // 1 to select below cubelet
            input left,             // 1 to select cubelet to the left
            input right,            // 1 to select cubelet to the right
```

```
        input enter,
        input colour_change,    // 1 to cycle colour
        input [10:0] hcount,    // horizontal index of current pixel (0..1023)
        input [9:0] vcount,     // vertical index of current pixel (0..767)
        input hsync,            // XVGA horizontal sync signal (active low)
        input vsync,            // XVGA vertical sync signal (active low)
        input blank,            // XVGA blanking (1 means output black pixel)
        output [0:161] cube_config_packed,
        output phsync,          // Input_State's horizontal sync
        output pvsync,          // Input_State's vertical sync
        output pblank,          // Input_State's blanking
        output [23:0]  pixel    // Input_State's pixel  // r=23:16, g=15:8, b=7:0
);

 reg initialise;
 reg pressed;
 reg [5:0] current_face;
 reg [2:0] cube_config [0:53];

 reg [10:0] xselected;
 reg [9:0] yselected;

 wire [23:0] greenpixel,pixelg1,pixelg2,pixelg3,pixelg4,pixelg5,pixelg6,pixelg7,pixelg8,
 pixelg9;
 wire [23:0] redpixel,pixelr1,pixelr2,pixelr3,pixelr4,pixelr5,pixelr6,pixelr7,pixelr8,pixelr9;
 wire [23:0] whitepixel,pixelw1,pixelw2,pixelw3,pixelw4,pixelw5,pixelw6,pixelw7,pixelw8,
 pixelw9;
 wire [23:0] orangepixel,pixelo1,pixelo2,pixelo3,pixelo4,pixelo5,pixelo6,pixelo7,pixelo8,
 pixelo9;
 wire [23:0] bluepixel,pixelb1,pixelb2,pixelb3,pixelb4,pixelb5,pixelb6,pixelb7,pixelb8,
 pixelb9;
 wire [23:0] yellowpixel,pixely1,pixely2,pixely3,pixely4,pixely5,pixely6,pixely7,pixely8,
 pixely9;
 wire [23:0] selected_pixel;

 genvar pk_idx;
 generate for (pk_idx=0; pk_idx<(54); pk_idx=pk_idx+1) begin:CUBE_ARRAY
 //assign cube_config_packed[((3)*pk_idx+((3)-1)):((3)*pk_idx)] =
 cube_config[pk_idx][((3)-1):0];
 assign cube_config_packed[((3)*pk_idx):((3)*pk_idx+((3)-1))] = cube_config[pk_idx][((3)-1):0
 ];
 end
 endgenerate

 parameter cwidth = 50; //cubelet face parameters
 parameter cheight = 50;
 parameter xcubecenter = (1024/2)-((cwidth/2)+1); //define the centre of the screen
 parameter ycubecenter = (768/2)-((cheight/2)+1);
 parameter gap = 2;//gap between cubelet faces
 parameter xpos1 = xcubecenter - (4*(cwidth+gap)); //define standard x positions
 parameter xpos2 = xcubecenter - (3*(cwidth+gap));
 parameter xpos3 = xcubecenter - (2*(cwidth+gap));
 parameter xpos4 = xcubecenter - (1*(cwidth+gap));
```

```
    parameter xpos5 = xcubecenter;
    parameter xpos6 = xcubecenter + (1*(cwidth+gap));
    parameter xpos7 = xcubecenter + (2*(cwidth+gap));
    parameter xpos8 = xcubecenter + (3*(cwidth+gap));
    parameter xpos9 = xcubecenter + (4*(cwidth+gap));
    parameter ypos1 = ycubecenter - (5*(cheight+gap)) - (cheight/2); //define standard y
    positions
    parameter ypos2 = ycubecenter - (4*(cheight+gap)) - (cheight/2);
    parameter ypos3 = ycubecenter - (3*(cheight+gap)) - (cheight/2);
    parameter ypos4 = ycubecenter - (2*(cheight+gap)) - (cheight/2);
    parameter ypos5 = ycubecenter - (1*(cheight+gap)) - (cheight/2);
    parameter ypos6 = ycubecenter + (0*(cheight+gap)) - (cheight/2);
    parameter ypos7 = ycubecenter + (0*(cheight+gap)) + (cheight/2) + gap;
    parameter ypos8 = ycubecenter + (1*(cheight+gap)) + (cheight/2) + gap;
    parameter ypos9 = ycubecenter + (2*(cheight+gap)) + (cheight/2) + gap;
    parameter ypos10 = ycubecenter + (3*(cheight+gap)) + (cheight/2) + gap;
    parameter ypos11 = ycubecenter + (4*(cheight+gap)) + (cheight/2) + gap;
    parameter ypos12 = ycubecenter + (5*(cheight+gap)) + (cheight/2) + gap;


    parameter grey = 0;//define colours
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    parameter g1 = 0;
    parameter g2 = 1;
    parameter g3 = 2;
    parameter g4 = 3;
    parameter g5 = 4;
    parameter g6 = 5;
    parameter g7 = 6;
    parameter g8 = 7;
    parameter g9 = 8;

    parameter r1 = 9;
    parameter r2 = 10;
    parameter r3 = 11;
    parameter r4 = 12;
    parameter r5 = 13;
    parameter r6 = 14;
    parameter r7 = 15;
    parameter r8 = 16;
    parameter r9 = 17;

    parameter w1 = 18;
    parameter w2 = 19;
    parameter w3 = 20;
    parameter w4 = 21;
    parameter w5 = 22;
    parameter w6 = 23;
    parameter w7 = 24;
    parameter w8 = 25;
    parameter w9 = 26;

    parameter o1 = 27;
    parameter o2 = 28;
    parameter o3 = 29;
    parameter o4 = 30;
    parameter o5 = 31;
    parameter o6 = 32;
    parameter o7 = 33;
    parameter o8 = 34;
    parameter o9 = 35;

    parameter b1 = 36;
    parameter b2 = 37;
    parameter b3 = 38;
    parameter b4 = 39;
    parameter b5 = 40;
    parameter b6 = 41;
    parameter b7 = 42;
    parameter b8 = 43;
    parameter b9 = 44;

    parameter y1 = 45;
    parameter y2 = 46;
    parameter y3 = 47;
    parameter y4 = 48;
    parameter y5 = 49;
    parameter y6 = 50;
    parameter y7 = 51;
    parameter y8 = 52;
    parameter y9 = 53;

    always @(posedge vclock) begin
        if(~initialise) begin //initialise cubelet faces correctly
            cube_config[g1] <= grey;
            cube_config[g2] <= grey;
            cube_config[g3] <= grey;
            cube_config[g4] <= grey;
            cube_config[g5] <= green;
            cube_config[g6] <= grey;
            cube_config[g7] <= grey;
            cube_config[g8] <= grey;
            cube_config[g9] <= grey;

            cube_config[r1] <= grey;
            cube_config[r2] <= grey;
            cube_config[r3] <= grey;
            cube_config[r4] <= grey;
            cube_config[r5] <= red;
            cube_config[r6] <= grey;
            cube_config[r7] <= grey;
            cube_config[r8] <= grey;
            cube_config[r9] <= grey;

            cube_config[w1] <= grey;
            cube_config[w2] <= grey;
            cube_config[w3] <= grey;
            cube_config[w4] <= grey;
            cube_config[w5] <= white;
            cube_config[w6] <= grey;
            cube_config[w7] <= grey;
            cube_config[w8] <= grey;
            cube_config[w9] <= grey;

            cube_config[o1] <= grey;
            cube_config[o2] <= grey;
            cube_config[o3] <= grey;
            cube_config[o4] <= grey;
            cube_config[o5] <= orange;
            cube_config[o6] <= grey;
            cube_config[o7] <= grey;
            cube_config[o8] <= grey;
            cube_config[o9] <= grey;

            cube_config[b1] <= grey;
            cube_config[b2] <= grey;
            cube_config[b3] <= grey;
            cube_config[b4] <= grey;
            cube_config[b5] <= blue;
            cube_config[b6] <= grey;
            cube_config[b7] <= grey;
            cube_config[b8] <= grey;
            cube_config[b9] <= grey;

            cube_config[y1] <= grey;
            cube_config[y2] <= grey;
            cube_config[y3] <= grey;
            cube_config[y4] <= grey;
            cube_config[y5] <= yellow;
            cube_config[y6] <= grey;
            cube_config[y7] <= grey;
            cube_config[y8] <= grey;
            cube_config[y9] <= grey;

            xselected <= xpos4;//initialise selection box on first cubelet face
            yselected <= ypos1;
            current_face <= 0;
            initialise <= 1;
        end
        else begin
            if (reset) initialise <= 0; //re-initialise when reset is pressed
            else case(current_face) //navigation case statement (one for each cubelet face)

                g1:     begin
                            xselected <= xpos4;
                            yselected <= ypos1;
                   //only  cycle one colour on every button press
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[g1] < 6) cube_config[g1] <= cube_config[g1]+1;
                                    else cube_config[g1] <= 1;
                                end
                                else if(up) begin
                                    current_face <= y7;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= g4;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= g2;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= g3;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                g2:     begin
                            xselected <= xpos5;
                            yselected <= ypos1;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[g2] < 6) cube_config[g2] <= cube_config[g2]+1;
                                    else cube_config[g2] <= 1;
                                end
                                else if(up) begin
                                    current_face <= y8;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= g8;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= g3;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= g1;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                g3:     begin
                            xselected <= xpos6;
                            yselected <= ypos1;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[g3] < 6) cube_config[g3] <= cube_config[g3]+1;
                                    else cube_config[g3] <= 1;
                                end
                                else if(up) begin
                                    current_face <= y9;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= g6;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= g1;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= g2;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                g4:     begin
                            xselected <= xpos4;
                            yselected <= ypos2;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[g4] < 6) cube_config[g4] <= cube_config[g4]+1;
                                    else cube_config[g4] <= 1;
                                end
                                else if(up) begin
```

```
                        current_face <= g1;
                        pressed<=1;
                    end
                    else if(down) begin
                        current_face <= g7;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= g6;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= g6;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                <= 0;

            end

        g6:     begin
                xselected <= xpos6;
                yselected <= ypos2;
                if(~pressed)begin
                    if(colour_change) begin
                        pressed <=1;
                        if(cube_config[g6] < 6) cube_config[g6] <= cube_config[g6]+1;
                        else cube_config[g6] <= 1;
                    end
                    else if(up) begin
                        current_face <= g3;
                        pressed<=1;
                    end
                    else if(down) begin
                        current_face <= g9;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= g4;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= g4;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                <= 0;

            end

        g7:     begin
```

```
                        xselected <= xpos4;
                        yselected <= ypos3;
                        if(~pressed)begin
                            if(colour_change) begin
                                pressed <=1;
                                if(cube_config[g7] < 6) cube_config[g7] <= cube_config[g7]+1;
                                else cube_config[g7] <= 1;
                            end
                            else if(up) begin
                                current_face <= g4;
                                pressed<=1;
                            end
                            else if(down) begin
                                current_face <= w1;
                                pressed<=1;
                            end
                            else if(right) begin
                                current_face <= g8;
                                pressed<=1;
                            end
                            else if(left) begin
                                current_face <= g9;
                                pressed<=1;
                            end
                        end
                        else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                        <= 0;

                    end

            g8:     begin
                        xselected <= xpos5;
                        yselected <= ypos3;
                        if(~pressed)begin
                            if(colour_change) begin
                                pressed <=1;
                                if(cube_config[g8] < 6) cube_config[g8] <= cube_config[g8]+1;
                                else cube_config[g8] <= 1;
                            end
                            else if(up) begin
                                current_face <= g2;
                                pressed<=1;
                            end
                            else if(down) begin
                                current_face <= w2;
                                pressed<=1;
                            end
                            else if(right) begin
                                current_face <= g9;
                                pressed<=1;
                            end
                            else if(left) begin
                                current_face <= g7;
                                pressed<=1;
                            end
                        end
                        else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                        <= 0;

                    end

            g9:     begin
                        xselected <= xpos6;
                        yselected <= ypos3;
                        if(~pressed)begin
                            if(colour_change) begin
                                pressed <=1;
                                if(cube_config[g9] < 6) cube_config[g9] <= cube_config[g9]+1;
                                else cube_config[g9] <= 1;
                            end
                            else if(up) begin
                                current_face <= g6;
                                pressed<=1;
                            end
                            else if(down) begin
                                current_face <= w3;
                                pressed<=1;
                            end
                            else if(right) begin
                                current_face <= g7;
                                pressed<=1;
                            end
                            else if(left) begin
                                current_face <= g8;
                                pressed<=1;
                            end
                        end
                        else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                        <= 0;

                    end

            r1:     begin
                        xselected <= xpos1;
                        yselected <= ypos4;
                        if(~pressed)begin
                            if(colour_change) begin
                                pressed <=1;
                                if(cube_config[r1] < 6) cube_config[r1] <= cube_config[r1]+1;
                                else cube_config[r1] <= 1;
                            end
                            else if(up) begin
                                current_face <= r7;
                                pressed<=1;
                            end
                            else if(down) begin
```

```
                        current_face <= r4;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= r2;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= o3;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                <= 0;

            end

    r2:     begin
                xselected <= xpos2;
                yselected <= ypos4;
                if(~pressed)begin
                    if(colour_change) begin
                        pressed <=1;
                        if(cube_config[r2] < 6) cube_config[r2] <= cube_config[r2]+1;
                        else cube_config[r2] <= 1;
                    end
                    else if(up) begin
                        current_face <= r8;
                        pressed<=1;
                    end
                    else if(down) begin
                        current_face <= r8;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= r3;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= r1;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                <= 0;

            end

    r3:     begin
                xselected <= xpos3;
                yselected <= ypos4;
                if(~pressed)begin
                    if(colour_change) begin
```

```
                        pressed <=1;
                        if(cube_config[r3] < 6) cube_config[r3] <= cube_config[r3]+1;
                        else cube_config[r3] <= 1;
                    end
                    else if(up) begin
                        current_face <= r9;
                        pressed<=1;
                    end
                    else if(down) begin
                        current_face <= r6;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= w1;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= r2;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                <= 0;

            end

        r4:     begin
                xselected <= xpos1;
                yselected <= ypos5;
                if(~pressed)begin
                    if(colour_change) begin
                        pressed <=1;
                        if(cube_config[r4] < 6) cube_config[r4] <= cube_config[r4]+1;
                        else cube_config[r4] <= 1;
                    end
                    else if(up) begin
                        current_face <= r1;
                        pressed<=1;
                    end
                    else if(down) begin
                        current_face <= r7;
                        pressed<=1;
                    end
                    else if(right) begin
                        current_face <= r6;
                        pressed<=1;
                    end
                    else if(left) begin
                        current_face <= o6;
                        pressed<=1;
                    end
                end
                else if(~colour_change && ~up && ~down && ~right && ~left) pressed
```

```
                    <= 0;

                end

        r6:     begin
                    xselected <= xpos3;
                    yselected <= ypos5;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[r6] < 6) cube_config[r6] <= cube_config[r6]+1;
                            else cube_config[r6] <= 1;
                        end
                        else if(up) begin
                            current_face <= r3;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= r9;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w4;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r4;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        r7:     begin
                    xselected <= xpos1;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[r7] < 6) cube_config[r7] <= cube_config[r7]+1;
                            else cube_config[r7] <= 1;
                        end
                        else if(up) begin
                            current_face <= r4;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= r1;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= r8;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o9;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        r8:     begin
                    xselected <= xpos2;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[r8] < 6) cube_config[r8] <= cube_config[r8]+1;
                            else cube_config[r8] <= 1;
                        end
                        else if(up) begin
                            current_face <= r2;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= r2;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= r9;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r7;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        r9:     begin
                    xselected <= xpos3;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[r9] < 6) cube_config[r9] <= cube_config[r9]+1;
                            else cube_config[r9] <= 1;
                        end
```

```
                        else if(up) begin
                            current_face <= r6;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= r3;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w7;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r8;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w1:     begin
                    xselected <= xpos4;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w1] < 6) cube_config[w1] <= cube_config[w1]+1;
                            else cube_config[w1] <= 1;
                        end
                        else if(up) begin
                            current_face <= g7;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= w4;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w2;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r3;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end
        w2:     begin
                    xselected <= xpos5;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w2] < 6) cube_config[w2] <= cube_config[w2]+1;
                            else cube_config[w2] <= 1;
                        end
                        else if(up) begin
                            current_face <= g8;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= w8;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w3;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w1;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w3:     begin
                    xselected <= xpos6;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w3] < 6) cube_config[w3] <= cube_config[w3]+1;
                            else cube_config[w3] <= 1;
                        end
                        else if(up) begin
                            current_face <= g9;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= w6;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o1;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w2;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w4:     begin
                    xselected <= xpos4;
                    yselected <= ypos5;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w4] < 6) cube_config[w4] <= cube_config[w4]+1;
                            else cube_config[w4] <= 1;
                        end
                        else if(up) begin
                            current_face <= w1;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= w7;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w6;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r6;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w6:     begin
                    xselected <= xpos6;
                    yselected <= ypos5;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w6] < 6) cube_config[w6] <= cube_config[w6]+1;
                            else cube_config[w6] <= 1;
                        end
                        else if(up) begin
                            current_face <= w3;
                            pressed<=1;
                        end
```

```
                        else if(down) begin
                            current_face <= w9;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o4;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w4;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w7:     begin
                    xselected <= xpos4;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[w7] < 6) cube_config[w7] <= cube_config[w7]+1;
                            else cube_config[w7] <= 1;
                        end
                        else if(up) begin
                            current_face <= w4;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= b1;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= w8;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= r9;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        w8:     begin
                    xselected <= xpos5;
                    yselected <= ypos6;
                    if(~pressed)begin
```

```
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[w8] < 6) cube_config[w8] <= cube_config[w8]+1;
                                    else cube_config[w8] <= 1;
                                end
                                else if(up) begin
                                    current_face <= w2;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= b2;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= w9;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= w7;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                w9:     begin
                            xselected <= xpos6;
                            yselected <= ypos6;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[w9] < 6) cube_config[w9] <= cube_config[w9]+1;
                                    else cube_config[w9] <= 1;
                                end
                                else if(up) begin
                                    current_face <= w6;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= b3;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= o7;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= w8;
                                    pressed<=1;
                                end
                            end
```

```
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        o1:     begin
                    xselected <= xpos7;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o1] < 6) cube_config[o1] <= cube_config[o1]+1;
                            else cube_config[o1] <= 1;
                        end
                        else if(up) begin
                            current_face <= o7;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o4;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o2;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w3;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        o2:     begin
                    xselected <= xpos8;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o2] < 6) cube_config[o2] <= cube_config[o2]+1;
                            else cube_config[o2] <= 1;
                        end
                        else if(up) begin
                            current_face <= o8;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o8;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o3;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o1;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

            o3:     begin
                    xselected <= xpos9;
                    yselected <= ypos4;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o3] < 6) cube_config[o3] <= cube_config[o3]+1;
                            else cube_config[o3] <= 1;
                        end
                        else if(up) begin
                            current_face <= o9;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o6;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= r1;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o2;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

            o4:     begin
                    xselected <= xpos7;
                    yselected <= ypos5;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o4] < 6) cube_config[o4] <= cube_config[o4]+1;
                            else cube_config[o4] <= 1;
                        end
                        else if(up) begin
                            current_face <= o1;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o7;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o6;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w6;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        o6:     begin
                    xselected <= xpos9;
                    yselected <= ypos5;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o6] < 6) cube_config[o6] <= cube_config[o6]+1;
                            else cube_config[o6] <= 1;
                        end
                        else if(up) begin
                            current_face <= o3;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o9;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= r4;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o4;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end
        o7:     begin
                    xselected <= xpos7;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o7] < 6) cube_config[o7] <= cube_config[o7]+1;
                            else cube_config[o7] <= 1;
                        end
                        else if(up) begin
                            current_face <= o4;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o1;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o8;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= w9;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        o8:     begin
                    xselected <= xpos8;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o8] < 6) cube_config[o8] <= cube_config[o8]+1;
                            else cube_config[o8] <= 1;
                        end
                        else if(up) begin
                            current_face <= o2;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o2;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= o9;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o7;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        o9:     begin
                    xselected <= xpos9;
                    yselected <= ypos6;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[o9] < 6) cube_config[o9] <= cube_config[o9]+1;
                            else cube_config[o9] <= 1;
                        end
                        else if(up) begin
                            current_face <= o6;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= o3;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= r7;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= o8;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        b1:     begin
                    xselected <= xpos4;
                    yselected <= ypos7;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[b1] < 6) cube_config[b1] <= cube_config[b1]+1;
                            else cube_config[b1] <= 1;
                        end
                        else if(up) begin
                            current_face <= w7;
                            pressed<=1;
```

```
                                end
                                else if(down) begin
                                    current_face <= b4;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= b2;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= b3;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                b2:     begin
                            xselected <= xpos5;
                            yselected <= ypos7;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[b2] < 6) cube_config[b2] <= cube_config[b2]+1;
                                    else cube_config[b2] <= 1;
                                end
                                else if(up) begin
                                    current_face <= w8;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= b8;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= b3;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= b1;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                b3:     begin
                            xselected <= xpos6;
                            yselected <= ypos7;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[b3] < 6) cube_config[b3] <= cube_config[b3]+1;
                                    else cube_config[b3] <= 1;
                                end
                                else if(up) begin
                                    current_face <= w9;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= b6;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= b1;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= b2;
                                    pressed<=1;
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                b4:     begin
                            xselected <= xpos4;
                            yselected <= ypos8;
                            if(~pressed)begin
                                if(colour_change) begin
                                    pressed <=1;
                                    if(cube_config[b4] < 6) cube_config[b4] <= cube_config[b4]+1;
                                    else cube_config[b4] <= 1;
                                end
                                else if(up) begin
                                    current_face <= b1;
                                    pressed<=1;
                                end
                                else if(down) begin
                                    current_face <= b7;
                                    pressed<=1;
                                end
                                else if(right) begin
                                    current_face <= b6;
                                    pressed<=1;
                                end
                                else if(left) begin
                                    current_face <= b6;
                                    pressed<=1;
                                end
```

```
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

            b6:     begin
                    xselected <= xpos6;
                    yselected <= ypos8;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[b6] < 6) cube_config[b6] <= cube_config[b6]+1;
                            else cube_config[b6] <= 1;
                        end
                        else if(up) begin
                            current_face <= b3;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= b9;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= b4;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= b4;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

            b7:     begin
                    xselected <= xpos4;
                    yselected <= ypos9;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[b7] < 6) cube_config[b7] <= cube_config[b7]+1;
                            else cube_config[b7] <= 1;
                        end
                        else if(up) begin
                            current_face <= b4;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y1;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= b8;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= b9;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        b8:     begin
                    xselected <= xpos5;
                    yselected <= ypos9;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[b8] < 6) cube_config[b8] <= cube_config[b8]+1;
                            else cube_config[b8] <= 1;
                        end
                        else if(up) begin
                            current_face <= b2;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y2;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= b9;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= b7;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        b9:     begin
                    xselected <= xpos6;
                    yselected <= ypos9;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[b9] < 6) cube_config[b9] <= cube_config[b9]+1;
                            else cube_config[b9] <= 1;
                        end
                        else if(up) begin
                            current_face <= b6;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y3;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= b7;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= b8;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y1:     begin
                    xselected <= xpos4;
                    yselected <= ypos10;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y1] < 6) cube_config[y1] <= cube_config[y1]+1;
                            else cube_config[y1] <= 1;
                        end
                        else if(up) begin
                            current_face <= b7;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y4;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y2;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y3;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;
                end

        y2:     begin
                    xselected <= xpos5;
                    yselected <= ypos10;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y2] < 6) cube_config[y2] <= cube_config[y2]+1;
                            else cube_config[y2] <= 1;
                        end
                        else if(up) begin
                            current_face <= b8;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y8;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y3;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y1;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y3:     begin
                    xselected <= xpos6;
                    yselected <= ypos10;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y3] < 6) cube_config[y3] <= cube_config[y3]+1;
                            else cube_config[y3] <= 1;
                        end
                        else if(up) begin
                            current_face <= b9;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y6;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y1;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y2;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y4:     begin
                    xselected <= xpos4;
                    yselected <= ypos11;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y4] < 6) cube_config[y4] <= cube_config[y4]+1;
                            else cube_config[y4] <= 1;
                        end
                        else if(up) begin
                            current_face <= y1;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y7;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y6;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y6;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y6:     begin
                    xselected <= xpos6;
                    yselected <= ypos11;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y6] < 6) cube_config[y6] <= cube_config[y6]+1;
                            else cube_config[y6] <= 1;
                        end
                        else if(up) begin
                            current_face <= y3;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= y9;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y4;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y4;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y7:     begin
                    xselected <= xpos4;
                    yselected <= ypos12;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y7] < 6) cube_config[y7] <= cube_config[y7]+1;
                            else cube_config[y7] <= 1;
                        end
                        else if(up) begin
                            current_face <= y4;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= g1;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y8;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y9;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

        y8:     begin
                    xselected <= xpos5;
```

```
                    yselected <= ypos12;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y8] < 6) cube_config[y8] <= cube_config[y8]+1;
                            else cube_config[y8] <= 1;
                        end
                        else if(up) begin
                            current_face <= y2;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= g2;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y9;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y7;
                            pressed<=1;
                        end
                    end
                    else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                    <= 0;

                end

            y9:     begin
                    xselected <= xpos6;
                    yselected <= ypos12;
                    if(~pressed)begin
                        if(colour_change) begin
                            pressed <=1;
                            if(cube_config[y9] < 6) cube_config[y9] <= cube_config[y9]+1;
                            else cube_config[y9] <= 1;
                        end
                        else if(up) begin
                            current_face <= y6;
                            pressed<=1;
                        end
                        else if(down) begin
                            current_face <= g3;
                            pressed<=1;
                        end
                        else if(right) begin
                            current_face <= y7;
                            pressed<=1;
                        end
                        else if(left) begin
                            current_face <= y8;
                            pressed<=1;
```

```
                                end
                            end
                            else if(~colour_change && ~up && ~down && ~right && ~left) pressed
                            <= 0;

                        end

                default: current_face <= g1;

            endcase

        end

    end

    //create green face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g1(.x(xpos4),.hcount(hcount),.y(ypos1),.vcount(vcount),.colour(cube_config[g1]),.
        pixel(pixelg1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g2(.x(xpos5),.hcount(hcount),.y(ypos1),.vcount(vcount),.colour(cube_config[g2]),.
        pixel(pixelg2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g3(.x(xpos6),.hcount(hcount),.y(ypos1),.vcount(vcount),.colour(cube_config[g3]),.
        pixel(pixelg3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g4(.x(xpos4),.hcount(hcount),.y(ypos2),.vcount(vcount),.colour(cube_config[g4]),.
        pixel(pixelg4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g5(.x(xpos5),.hcount(hcount),.y(ypos2),.vcount(vcount),.colour(cube_config[g5]),.
        pixel(pixelg5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g6(.x(xpos6),.hcount(hcount),.y(ypos2),.vcount(vcount),.colour(cube_config[g6]),.
        pixel(pixelg6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g7(.x(xpos4),.hcount(hcount),.y(ypos3),.vcount(vcount),.colour(cube_config[g7]),.
        pixel(pixelg7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g8(.x(xpos5),.hcount(hcount),.y(ypos3),.vcount(vcount),.colour(cube_config[g8]),.
        pixel(pixelg8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_g9(.x(xpos6),.hcount(hcount),.y(ypos3),.vcount(vcount),.colour(cube_config[g9]),.
        pixel(pixelg9));


    //create red face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r1(.x(xpos1),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[r1]),.
        pixel(pixelr1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r2(.x(xpos2),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[r2]),.
        pixel(pixelr2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
```

```
        blob_r3(.x(xpos3),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[r3]),.
        pixel(pixelr3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r4(.x(xpos1),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[r4]),.
        pixel(pixelr4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r5(.x(xpos2),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[r5]),.
        pixel(pixelr5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r6(.x(xpos3),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[r6]),.
        pixel(pixelr6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r7(.x(xpos1),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[r7]),.
        pixel(pixelr7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r8(.x(xpos2),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[r8]),.
        pixel(pixelr8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_r9(.x(xpos3),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[r9]),.
        pixel(pixelr9));


    //create white face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w1(.x(xpos4),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[w1]),.
        pixel(pixelw1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w2(.x(xpos5),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[w2]),.
        pixel(pixelw2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w3(.x(xpos6),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[w3]),.
        pixel(pixelw3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w4(.x(xpos4),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[w4]),.
        pixel(pixelw4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w5(.x(xpos5),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[w5]),.
        pixel(pixelw5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w6(.x(xpos6),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[w6]),.
        pixel(pixelw6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w7(.x(xpos4),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[w7]),.
        pixel(pixelw7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w8(.x(xpos5),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[w8]),.
        pixel(pixelw8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_w9(.x(xpos6),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[w9]),.
        pixel(pixelw9));


    //create orange face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o1(.x(xpos7),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[o1]),.
        pixel(pixelo1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o2(.x(xpos8),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[o2]),.
        pixel(pixelo2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o3(.x(xpos9),.hcount(hcount),.y(ypos4),.vcount(vcount),.colour(cube_config[o3]),.
        pixel(pixelo3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o4(.x(xpos7),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[o4]),.
        pixel(pixelo4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o5(.x(xpos8),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[o5]),.
        pixel(pixelo5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o6(.x(xpos9),.hcount(hcount),.y(ypos5),.vcount(vcount),.colour(cube_config[o6]),.
        pixel(pixelo6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o7(.x(xpos7),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[o7]),.
        pixel(pixelo7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o8(.x(xpos8),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[o8]),.
        pixel(pixelo8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_o9(.x(xpos9),.hcount(hcount),.y(ypos6),.vcount(vcount),.colour(cube_config[o9]),.
        pixel(pixelo9));


    //create blue face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b1(.x(xpos4),.hcount(hcount),.y(ypos7),.vcount(vcount),.colour(cube_config[b1]),.
        pixel(pixelb1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b2(.x(xpos5),.hcount(hcount),.y(ypos7),.vcount(vcount),.colour(cube_config[b2]),.
        pixel(pixelb2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b3(.x(xpos6),.hcount(hcount),.y(ypos7),.vcount(vcount),.colour(cube_config[b3]),.
        pixel(pixelb3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b4(.x(xpos4),.hcount(hcount),.y(ypos8),.vcount(vcount),.colour(cube_config[b4]),.
        pixel(pixelb4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b5(.x(xpos5),.hcount(hcount),.y(ypos8),.vcount(vcount),.colour(cube_config[b5]),.
        pixel(pixelb5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b6(.x(xpos6),.hcount(hcount),.y(ypos8),.vcount(vcount),.colour(cube_config[b6]),.
        pixel(pixelb6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b7(.x(xpos4),.hcount(hcount),.y(ypos9),.vcount(vcount),.colour(cube_config[b7]),.
        pixel(pixelb7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b8(.x(xpos5),.hcount(hcount),.y(ypos9),.vcount(vcount),.colour(cube_config[b8]),.
        pixel(pixelb8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_b9(.x(xpos6),.hcount(hcount),.y(ypos9),.vcount(vcount),.colour(cube_config[b9]),.
        pixel(pixelb9));



    //create yellow face
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y1(.x(xpos4),.hcount(hcount),.y(ypos10),.vcount(vcount),.colour(cube_config[y1]),.
        pixel(pixely1));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y2(.x(xpos5),.hcount(hcount),.y(ypos10),.vcount(vcount),.colour(cube_config[y2]),.
        pixel(pixely2));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y3(.x(xpos6),.hcount(hcount),.y(ypos10),.vcount(vcount),.colour(cube_config[y3]),.
        pixel(pixely3));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y4(.x(xpos4),.hcount(hcount),.y(ypos11),.vcount(vcount),.colour(cube_config[y4]),.
        pixel(pixely4));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y5(.x(xpos5),.hcount(hcount),.y(ypos11),.vcount(vcount),.colour(cube_config[y5]),.
        pixel(pixely5));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y6(.x(xpos6),.hcount(hcount),.y(ypos11),.vcount(vcount),.colour(cube_config[y6]),.
        pixel(pixely6));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y7(.x(xpos4),.hcount(hcount),.y(ypos12),.vcount(vcount),.colour(cube_config[y7]),.
        pixel(pixely7));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y8(.x(xpos5),.hcount(hcount),.y(ypos12),.vcount(vcount),.colour(cube_config[y8]),.
        pixel(pixely8));
    blob #(.WIDTH(cwidth),.HEIGHT(cheight))
        blob_y9(.x(xpos6),.hcount(hcount),.y(ypos12),.vcount(vcount),.colour(cube_config[y9]),.
        pixel(pixely9));

    assign greenpixel = pixelg1 | pixelg2 | pixelg3 | pixelg4 | pixelg5 | pixelg6 | pixelg7 |
    pixelg8 | pixelg9;
    assign redpixel = pixelr1 | pixelr2 | pixelr3 | pixelr4 | pixelr5 | pixelr6 | pixelr7 |
    pixelr8 | pixelr9;
    assign whitepixel = pixelw1 | pixelw2 | pixelw3 | pixelw4 | pixelw5 | pixelw6 | pixelw7 |
    pixelw8 | pixelw9;
    assign orangepixel = pixelo1 | pixelo2 | pixelo3 | pixelo4 | pixelo5 | pixelo6 | pixelo7 |
    pixelo8 | pixelo9;
    assign bluepixel = pixelb1 | pixelb2 | pixelb3 | pixelb4 | pixelb5 | pixelb6 | pixelb7 |
    pixelb8 | pixelb9;
    assign yellowpixel = pixely1 | pixely2 | pixely3 | pixely4 | pixely5 | pixely6 | pixely7 |
    pixely8 | pixely9;

    //create selection box
    hollow selected(.x(xselected),.hcount(hcount),.y(yselected),.vcount(vcount),.pixel(
    selected_pixel));

    assign pixel = selected_pixel ? selected_pixel : (greenpixel | redpixel | whitepixel |
    orangepixel | bluepixel | yellowpixel);


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

endmodule



////////////////////////////////////////////////////////////////////////
// Description: Generate rectangle on screen
////////////////////////////////////////////////////////////////////////

module blob
   #(parameter WIDTH = 64,            // default width: 64 pixels
               HEIGHT = 64)           // default height: 64 pixels
   (input [10:0] x,hcount,
    input [9:0] y,vcount,
     input [2:0] colour,
    output reg [23:0] pixel);

    parameter grey = 0;
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    reg [23:0] COLOR;

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

        case(colour)

            grey: COLOR = 24'hAAAAAA;
            red: COLOR = 24'hFF0000;
            orange: COLOR = 24'hFF8000;
            yellow: COLOR = 24'hFFFF00;
            green: COLOR = 24'h00FF00;
            blue: COLOR = 24'h0000FF;
            white: COLOR = 24'hFFFFFF;
            black: COLOR = 24'h000000;
            default: COLOR = 24'h000000;

        endcase

   end
```

```
endmodule


///////////////////////////////////////////////////////////////////////////////
// Author: Jack
///////////////////////////////////////////////////////////////////////////////


module hollow
   #(parameter WIDTH = 50,      // default width: 50 pixels
     THICKNESS = 4,             //default thickness: 4 pixels
     COLOR = 24'hFF1FFF)        // default colour: pink
   (input [10:0] x,hcount,
    input [9:0] y,vcount,
     output reg [23:0] pixel);

    reg sides;
    reg top;
    reg bottom;

    always @ * begin
      if ((((hcount >= (x-THICKNESS+1)) && (hcount < (x+1))) || ((hcount >= (x+WIDTH-1)) && (
      hcount < (x+WIDTH + THICKNESS - 1)))) && (vcount >= y+1 && vcount < (y+WIDTH-1))) sides =
      1;
      else sides = 0;
        if ((hcount >= (x-THICKNESS+1) && hcount < (x+WIDTH+THICKNESS-1)) && (vcount >= (y-
        THICKNESS+1) && vcount < (y+1))) top = 1;
      else top = 0;
        if ((hcount >= (x-THICKNESS+1) && hcount < (x+WIDTH+THICKNESS-1)) && (vcount >= (y+WIDTH
        -1) && vcount < (y+WIDTH+THICKNESS-1))) bottom = 1;
      else bottom = 0;
        if(top||bottom||sides)pixel=COLOR;
        else pixel = 0;

   end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Track the progression of two state machines.  One state machine
// (the verified state machine) represents whether the configuration inputted by the
// user is valid.  The second state machine (the algorithm state machine) tracks which
// cubelets have been correctly positioned in the process of solving the cube.
//////////////////////////////////////////////////////////////////////////////////

module Solution_State_Machine(
    input clk,                              // clock
    input reset,                            // reset
    input start,                            // goes high to trigger solving
    input [0:161] start_cube_config,        // starting cube configuration
    input next,                             // goes high to trigger next rotation
    output reg [0:161] cube_config,         // current cube configuration
    output [4:0] animation_type,            // absolute rotation to occur in animation
    output reg [4:0] state,                 // current algorithm state machine state
    output reg [2:0] run_state,             // current verified state machine state
    output clock_pulse,                     // clock square wave for sending data to other labkit
    output sync_pulse,                      // sync pulse for sending data to other labkit
    output serial_data                      // serial data being sent to other labkit
    );

    // Define parameters for each of the colors
    localparam n = 3'd0;    // no color
    localparam r = 3'd1;    // red
    localparam o = 3'd2;    // orange
    localparam y = 3'd3;    // yellow
    localparam g = 3'd4;    // green
    localparam b = 3'd5;    // blue
    localparam w = 3'd6;    // white

    // Allow the orientation of the cube to be respecified in order to generalize
    // the move sequences
    wire [0:17] orientation_colors;
    reg [0:17] orientation_colors_reg;
    assign orientation_colors = orientation_colors_reg;

    reg start_find; // Trigger a find operation

    // Set up first find cubelet module
    reg [0:8] cubelet_colors1;
    wire [0:8] face_positions1;
    wire find_done;
    wire fail1;

    Find_Cubelet find1 (
        .clk(clk),
        .start_find(start_find),
        .cubelet_colors(cubelet_colors1),
        .cube_config(cube_config),
        .orientation_colors(orientation_colors),
        .face_positions(face_positions1),
        .find_done(find_done),
        .fail(fail1)
    );

    // Set up second find cubelet module
    reg [0:8] cubelet_colors2;
    wire [0:8] face_positions2;
    wire find_done2;
    wire fail2;

    Find_Cubelet find2 (
        .clk(clk),
        .start_find(start_find),
        .cubelet_colors(cubelet_colors2),
        .cube_config(cube_config),
        .orientation_colors(orientation_colors),
        .face_positions(face_positions2),
        .find_done(find_done2),
        .fail(fail2)
    );

    // Set up third find cubelet module
    reg [0:8] cubelet_colors3;
    wire [0:8] face_positions3;
    wire find_done3;
    wire fail3;

    Find_Cubelet find3 (
        .clk(clk),
        .start_find(start_find),
        .cubelet_colors(cubelet_colors3),
        .cube_config(cube_config),
        .orientation_colors(orientation_colors),
        .face_positions(face_positions3),
        .find_done(find_done3),
        .fail(fail3)
    );

    // Set up fourth find cubelet module
    reg [0:8] cubelet_colors4;
    wire [0:8] face_positions4;
    wire find_done4;
    wire fail4;

    Find_Cubelet find4 (
        .clk(clk),
        .start_find(start_find),
        .cubelet_colors(cubelet_colors4),
        .cube_config(cube_config),
        .orientation_colors(orientation_colors),
        .face_positions(face_positions4),
        .find_done(find_done4),
        .fail(fail4)
    );

    // Set up rotation module
    reg [4:0] rotate_type;
    reg start_rotate;
    wire [0:161] new_cube_config;
    wire rotate_done;

    Rotate rotate (
        .clk(clk),
        .rotate_type(rotate_type),
        .cc(cube_config),
        .orientation_colors(orientation_colors),
        .start_rotate(start_rotate),
        .new_cube_config(new_cube_config),
        .animation_type(animation_type),
        .rotate_done(rotate_done)
    );

    // Set up module for sending rotation and cube configuration
    // information to the second labkit
    wire trigger;
    reg trigger_reg = 0;
    assign trigger = trigger_reg;
    Serial_send serial (.clock(clk),.cube_config(cube_config),
        .animation_type(animation_type),.trigger(trigger),
        .clock_pulse(clock_pulse),.sync_pulse(sync_pulse),
        .data(serial_data),.state_wire(state_wire));

    // Set up module for correcting the top layer edge cubelets
    wire [5:0] length_top_edge;
    wire [199:0] move_list_top_edge;
    wire fail_top_edge;
    Top_Face_Edge topedge (
        .face_positions(face_positions1),
        .move_list(move_list_top_edge),
        .num_moves(length_top_edge),
        .fail(fail_top_edge)
    );

    // Set up module for correcting the top layer corner cubelets
    wire [5:0] length_top_corner;
    wire [199:0] move_list_top_corner;
    wire fail_top_corner;
    Top_Face_Corner topcorner (
        .face_positions(face_positions1),
        .move_list(move_list_top_corner),
        .num_moves(length_top_corner),
        .fail(fail_top_corner)
    );

    // Set up module for correcting the middle layer edge cubelets
    wire [5:0] length_middle;
    wire[199:0] move_list_middle;
    wire fail_middle;
    Middle_Edge middle (
        .face_positions(face_positions1),
        .move_list(move_list_middle),
        .num_moves(length_middle),
        .fail(fail_middle)
    );

    // Set up module for correcting the positions of the bottom layer corner cubelets
    wire [5:0] length_bottom_corner_position;
    wire [199:0] move_list_bottom_corner_position;
    wire fail_bottom_corner_position;
    Bottom_Corner_Position bottomcornerposition (
        .RYBposition(face_positions1),
        .RWBposition(face_positions2),
        .OWBposition(face_positions3),
        .OYBposition(face_positions4),
        .move_list(move_list_bottom_corner_position),
        .num_moves(length_bottom_corner_position),
        .fail(fail_bottom_corner_position)
    );

    // Set up module for correcting the orientations of the bottom layer corner cubelets
    wire [5:0] length_bottom_corner_orientation;
    wire [199:0] move_list_bottom_corner_orientation;
    wire fail_bottom_corner_orientation;
    Bottom_Corner_Orientation bottomcornerorientation (
        .BRYposition(face_positions1),
        .BRWposition(face_positions2),
        .BOWposition(face_positions3),
        .BOYposition(face_positions4),
        .move_list(move_list_bottom_corner_orientation),
        .num_moves(length_bottom_corner_orientation),
        .fail(fail_bottom_corner_orientation)
    );

    // Set up module for correcting the positions of the bottom layer edge cubelets
    wire [5:0] length_bottom_edge_position;
    wire [199:0] move_list_bottom_edge_position;
    wire fail_bottom_edge_position;
    Bottom_Edge_Position bottomedgeposition (
        .RBposition(face_positions1),
        .WBposition(face_positions2),
        .move_list(move_list_bottom_edge_position),
        .num_moves(length_bottom_edge_position),
        .fail(fail_bottom_edge_position)
    );

    // Set up module for correcting the orientations of the bottom layer edge cubelets
    wire [5:0] length_bottom_edge_orientation;
    wire [199:0] move_list_bottom_edge_orientation;
    wire fail_bottom_edge_orientation;
    Bottom_Edge_Orientation bottomedgeorientation (
        .BRposition(face_positions1),
        .BWposition(face_positions2),
        .BOposition(face_positions3),
        .BYposition(face_positions4),
        .move_list(move_list_bottom_edge_orientation),
        .num_moves(length_bottom_edge_orientation),
        .fail(fail_bottom_edge_orientation)
    );

    reg [5:0] current;      // Track current progress through the present rotation sequence

    // Define each of the algorithm states
    localparam GY = 5'd1;       // Working to correct the green-yellow cubelet
    localparam GR = 5'd2;       // Working to correct the green-red cubelet
    localparam GW = 5'd3;       // Working to correct the green-white cubelet
    localparam GO = 5'd4;       // Working to correct the green-orange cubelet
    localparam GYR = 5'd5;      // Working to correct the green-yellow-red cubelet
    localparam GRW = 5'd6;      // Working to correct the green-red-white cubelet
    localparam GWO = 5'd7;      // Working to correct the green-white-orange cubelet
    localparam GOY = 5'd8;      // Working to correct the green-orange-yellow cubelet
    localparam YR = 5'd9;       // Working to correct the yellow-red cubelet
    localparam RW = 5'd10;      // Working to correct the red-white cubelet
    localparam WO = 5'd11;      // Working to correct the white-orange cubelet
    localparam OY = 5'd12;      // Working to correct the orange-yellow cubelet
    localparam BLCP = 5'd13;    // Working to correct the positions of the bottom layer corner
    cubelets
    localparam BLCO = 5'd14;    // Working to correct the orientations of the bottom layer
    corner cubelets
    localparam BLEP = 5'd15;    // Working to correct the positions of the bottom layer edge
    cubelets
    localparam BLEO = 5'd16;    // Working to correct the orientations of the bottom layer edge
    cubelets
    localparam DONE = 5'd17;    // Cube either solved or no solution possible

    // Create registers to be able to track if signals have changed
    reg prev_next;
    reg prev_start;
    reg prev_find_done;
    reg prev_find_done_combo;
    reg prev_rotate_done;
    reg prev_verify_next;

    reg [199:0] my_move_list;       // Rotation sequence for completing algorithm state
    reg ready = 0;                  // Ready for next rotation
    reg fail_step = 0;              // Goes high to indicate failure

    // Define each of the states for the verified state machine
    localparam not_verified = 3'd0;         // Configuration not yet verified
    localparam verifying = 3'd1;            // Working to check configuration
    localparam verified = 3'd2;             // Configuration deemed valid
    localparam pause = 3'd3;                // Wait for solution state to settle before doing
    first rotation
    localparam rotate_first = 3'd4;         // Trigger the first rotation
    localparam animation_running = 3'd5;    // Rely on the user to trigger all remaining
    rotations

    // Create registers to allow checking of starting cube configuration
    reg verify_next = 0;
    reg [7:0] verify_inner;
    reg [7:0] verify_rotate;
    reg [7:0] pause_delay;
    reg first_rotation = 0;

    // Track whether all find modules have completed their find operations
    wire find_done_combo;
    assign find_done_combo = find_done && find_done2 && find_done3 && find_done4;

    always @(posedge clk) begin
        // Store the status of the next, start, and done signals
        prev_next <= next;
        prev_verify_next <= verify_next;
        prev_start <= start;
        prev_find_done <= find_done;
        prev_find_done_combo <= find_done_combo;
        prev_rotate_done <= rotate_done;

        // If the system is reset, the starting configuration has not been verified
        if (reset) run_state <= not_verified;

        // If the start signal goes high, prepare to start solving the cube by going
        // to the starting algorithm state and clearing the start, current, and fail signals
        if (start != prev_start && start == 1) begin
            cube_config <= start_cube_config;
            start_find <= 0;
            current <= 0;
            state <= GY;
            ready <= 0;
            start_rotate <= 0;
            fail_step <= 0;
        end

        // Control the verified state machine
        case (run_state)

            // Transition to checking the starting configuration if the start signal goes high
            not_verified: begin
                if (start != prev_start && start == 1) begin
                    run_state <= verifying;
                    verify_inner <= 8'd1;
                    verify_rotate <= 8'd1;
                    verify_next <= 0;
                end
                end
            // Check whether the starting configuration is valid
            verifying: begin
                // Every 512 clock signals, send a next pulse to trigger the next rotation
                verify_inner <= verify_inner + 8'd1;
                if (verify_inner == 8'd0) begin
                    verify_next <= !verify_next;
                    // Allow up to 256 total rotations when checking the configuration
                    if (verify_next) verify_rotate <= verify_rotate + 8'd1;
                end
                // If the checking has terminated without failure, the configuration is valid
                if (state == DONE && !fail_step) run_state <= verified;
                // If failure has occured, the configuration must be fixed by the user
                if (fail_step == 1) run_state <= not_verified;
                // Update the state if the maximum number of rotations has been reached
                if (verify_rotate == 8'd0) run_state <= fail_step ? not_verified : verified;
                end

            // Wait until the user presses the start button again to start solving the cube
            verified: begin
                if (start != prev_start && start == 1) begin
                    run_state <= pause;
                    pause_delay <= 8'd1;
                end
                end

            // Wait 256 clock cycles to allow the solution state to settle before triggering
            // the first rotation
            pause: begin
                pause_delay <= pause_delay + 1;
                if (pause_delay == 8'd0) begin
                    first_rotation <= 1;
                    run_state <= rotate_first;
                end
                end

            // Trigger the first rotation
            rotate_first: begin
                first_rotation <= 0;
                run_state <= animation_running;
                end

            // Once in the animation_running state, rely on the user to trigger all remaining
            rotations
            animation_running: begin end

            // Default to the configuration needing to be verified
            default: run_state <= not_verified;
        endcase

        // Control the algorithm state machine
        case (state)
            // Fix the green-yellow cubelet
            GY: begin
```

```
                // Set the orientation with green on top and yellow in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,y,w,r,o};
                    cubelet_colors1 <= {g,y,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_edge;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_edge)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_edge) begin
                    current <= 0;
                    state <= GR;
                    ready <= 0;
                    start_find <= 0;
                end
                end
            // Fix the green-red cubelet
            GR: begin
                // Set the orientation with green on top and red in front, and locate the cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,r,o,w,y};
                    cubelet_colors1 <= {g,r,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_edge;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_edge)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_edge) begin
                    current <= 0;
                    state <= GW;
                    ready <= 0;
                    start_find <= 0;
```

```
        end
    end
// Fix the green-white cubelet
GW: begin
    // Set the orientation with green on top and white in front, and locate the
    cubelet
    if (!ready && !start_find && current == 0) begin
        orientation_colors_reg <= {g,b,w,y,o,r};
        cubelet_colors1 <= {g,w,n};
        start_find <= 1;
    end
    // When the cubelet is found, store the new list of rotations
    if (find_done != prev_find_done && find_done == 1) begin
        start_find <= 0;
        my_move_list <= move_list_top_edge;
        ready <= 1;
    end
    // If finding the cubelet or an appropriate move sequence has failed, go to the
    end state
    if (ready && (fail1 || fail_top_edge)) begin
        fail_step <= 1;
        state <= DONE;
    end
    // Go to the next state when the move sequence is completed
    if (ready && current == length_top_edge) begin
        current <= 0;
        state <= GO;
        ready <= 0;
        start_find <= 0;
    end
    end
// Fix the green-orange cubelet
GO: begin
    // Set the orientation with green on top and orange in front, and locate the
    cubelet
    if (!ready && !start_find && current == 0) begin
        orientation_colors_reg <= {g,b,o,r,y,w};
        cubelet_colors1 <= {g,o,n};
        start_find <= 1;
    end
    // When the cubelet is found, store the new list of rotations
    if (find_done != prev_find_done && find_done == 1) begin
        start_find <= 0;
        my_move_list <= move_list_top_edge;
        ready <= 1;
    end
    // If finding the cubelet or an appropriate move sequence has failed, go to the
    end state
    if (ready && (fail1 || fail_top_edge)) begin
        fail_step <= 1;
        state <= DONE;
    end
    // Go to the next state when the move sequence is completed
```

```
                if (ready && current == length_top_edge) begin
                    current <= 0;
                    state <= GYR;
                    ready <= 0;
                end
                end
            // Fix the green-yellow-red cubelet
            GYR: begin
                // Set the orientation with green on top and yellow in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,y,w,r,o};
                    cubelet_colors1 <= {g,y,r};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_corner;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_corner)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_corner) begin
                    current <= 0;
                    state <= GRW;
                    ready <= 0;
                end
                end
            // Fix the green-red-white cubelet
            GRW: begin
                // Set the orientation with green on top and red in front, and locate the cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,r,o,w,y};
                    cubelet_colors1 <= {g,r,w};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_corner;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_corner)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_corner) begin
                    current <= 0;
                    state <= GWO;
                    ready <= 0;
                end
                end
            // Fix the green-white-orange cubelet
            GWO: begin
                // Set the orientation with green on top and white in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,w,y,o,r};
                    cubelet_colors1 <= {g,w,o};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_corner;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_corner)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_corner) begin
                    current <= 0;
                    state <= GOY;
                    ready <= 0;
                end
                end
            // Fix the green-orange-yellow cubelet
            GOY: begin
                // Set the orientation with green on top and orange in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,o,r,y,w};
                    cubelet_colors1 <= {g,o,y};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_top_corner;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_top_corner)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_top_corner) begin
                    current <= 0;
                    state <= YR;
                    ready <= 0;
                end
                end
            // Fix the yellow-red cubelet
            YR: begin
                // Set the orientation with green on top and yellow in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,y,w,r,o};
                    cubelet_colors1 <= {y,r,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_middle;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_middle)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_middle) begin
                    current <= 0;
                    state <= RW;
                    ready <= 0;
                end
                end
            // Fix the red-white cubelet
            RW: begin
                // Set the orientation with green on top and red in front, and locate the cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,r,o,w,y};
                    cubelet_colors1 <= {r,w,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_middle;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_middle)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_middle) begin
                    current <= 0;
                    state <= WO;
                    ready <= 0;
                end
                end
            // Fix the white-orange cubelet
            WO: begin
                // Set the orientation with green on top and white in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,w,y,o,r};
                    cubelet_colors1 <= {w,o,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
                    my_move_list <= move_list_middle;
                    ready <= 1;
                end
                // If finding the cubelet or an appropriate move sequence has failed, go to the
                end state
                if (ready && (fail1 || fail_middle)) begin
                    fail_step <= 1;
                    state <= DONE;
                end
                // Go to the next state when the move sequence is completed
                if (ready && current == length_middle) begin
                    current <= 0;
                    state <= OY;
                    ready <= 0;
                end
                end
            // Fix the orange-yellow cubelet
            OY: begin
                // Set the orientation with green on top and orange in front, and locate the
                cubelet
                if (!ready && !start_find && current == 0) begin
                    orientation_colors_reg <= {g,b,o,r,y,w};
                    cubelet_colors1 <= {o,y,n};
                    start_find <= 1;
                end
                // When the cubelet is found, store the new list of rotations
                if (find_done != prev_find_done && find_done == 1) begin
                    start_find <= 0;
```

```
                my_move_list <= move_list_middle;
                ready <= 1;
            end
            // If finding the cubelet or an appropriate move sequence has failed, go to the
            end state
            if (ready && (fail1 || fail_middle)) begin
                fail_step <= 1;
                state <= DONE;
            end
            // Go to the next state when the move sequence is completed
            if (ready && current == length_middle) begin
                current <= 0;
                state <= BLCP;
                ready <= 0;
            end
            end
        // Fix the position of the bottom layer corner cubelets
        BLCP: begin
            // Set the orientation with green on top and red in front, and locate the four
            corner cubelets
            if (!ready && !start_find && current == 0) begin
                orientation_colors_reg <= {b,g,r,o,y,w};
                cubelet_colors1 <= {r,y,b};
                cubelet_colors2 <= {r,w,b};
                cubelet_colors3 <= {o,w,b};
                cubelet_colors4 <= {o,y,b};
                start_find <= 1;
            end
            // When the cubelet is found, store the new list of rotations
            if (find_done_combo != prev_find_done_combo && find_done_combo == 1) begin
                start_find <= 0;
                my_move_list <= move_list_bottom_corner_position;
                ready <= 1;
            end
            // If finding one or more of the cubelets or an appropriate move sequence has
            failed, go to the end state
            if (ready && (fail1 || fail2 || fail3 || fail4 || fail_bottom_corner_position))
            begin
                fail_step <= 1;
                state <= DONE;
            end
            // Go to the next state when the move sequence is completed
            if (ready && current == length_bottom_corner_position) begin
                current <= 0;
                state <= BLCO;
                ready <= 0;
            end
            end
        // Fix the orientation of the bottom layer corner cubelets
        BLCO: begin
            // Set the orientation with green on top and red in front, and locate the four
            corner cubelets
            if (!ready && !start_find && current == 0) begin
```

```
                orientation_colors_reg <= {b,g,r,o,y,w};
                cubelet_colors1 <= {b,r,y};
                cubelet_colors2 <= {b,r,w};
                cubelet_colors3 <= {b,o,w};
                cubelet_colors4 <= {b,o,y};
                start_find <= 1;
            end
            // When the cubelet is found, store the new list of rotations
            if (find_done_combo != prev_find_done_combo && find_done_combo == 1) begin
                start_find <= 0;
                my_move_list <= move_list_bottom_corner_orientation;
                ready <= 1;
            end
            // If finding one or more of the cubelets or an appropriate move sequence has
            failed, go to the end state
            if (ready && (fail1 || fail2 || fail3 || fail4 || fail_bottom_corner_orientation
            )) begin
                fail_step <= 1;
                state <= DONE;
            end
            // Go to the next state when the move sequence is completed
            if (ready && current == length_bottom_corner_orientation) begin
                current <= 0;
                state <= BLEP;
                ready <= 0;
            end
            end
        // Fix the position of the bottom layer edge cubelets
        BLEP: begin
            // Set the orientation with green on top and red in front, and locate two edge
            cubelets
            if (!ready && !start_find && current == 0) begin
                orientation_colors_reg <= {b,g,r,o,y,w};
                cubelet_colors1 <= {r,b,n};
                cubelet_colors2 <= {w,b,n};
                start_find <= 1;
            end
            // When the cubelet is found, store the new list of rotations
            if (find_done_combo != prev_find_done_combo && find_done_combo == 1) begin
                start_find <= 0;
                my_move_list <= move_list_bottom_edge_position;
                ready <= 1;
            end
            // If finding one or more of the cubelets or an appropriate move sequence has
            failed, go to the end state
            if (ready && (fail1 || fail2 || fail3 || fail4 || fail_bottom_edge_position))
            begin
                fail_step <= 1;
                state <= DONE;
            end
            // Go to the next state when the move sequence is completed
            if (ready && current == length_bottom_edge_position) begin
                current <= 0;
                state <= BLEO;
                ready <= 0;
            end
            end
        // Fix the orientation of the bottom layer edge cubelets
        BLEO: begin
            // Set the orientation with green on top and red in front, and locate the four
            edge cubelets
            if (!ready && !start_find && current == 0) begin
                orientation_colors_reg <= {b,g,r,o,y,w};
                cubelet_colors1 <= {b,r,n};
                cubelet_colors2 <= {b,w,n};
                cubelet_colors3 <= {b,o,n};
                cubelet_colors4 <= {b,y,n};
                start_find <= 1;
            end
            // When the cubelet is found, store the new list of rotations
            if (find_done_combo != prev_find_done_combo && find_done_combo == 1) begin
                start_find <= 0;
                my_move_list <= move_list_bottom_edge_orientation;
                ready <= 1;
            end
            // If finding one or more of the cubelets or an appropriate move sequence has
            failed, go to the end state
            if (ready && (fail1 || fail2 || fail3 || fail4 || fail_bottom_edge_orientation))
             begin
                fail_step <= 1;
                state <= DONE;
            end
            // Go to the next state when the move sequence is completed
            if (ready && current == length_bottom_edge_orientation) begin
                current <= 0;
                state <= DONE;
                ready <= 0;
            end
            end
        // Do nothing if in done state
        DONE: begin end
        // Default behavior is to go to the starting state
        default: begin state <= GY; end
    endcase

    // Trigger a rotation if the find operations are complete and either the verifying
    state machine or the user asks for a rotation
    if (ready == 1 && start_rotate == 0 && ((run_state == verifying && prev_verify_next !=
    verify_next && verify_next == 1) || (run_state == rotate_first && first_rotation == 1)
    || (run_state == animation_running && prev_next != next && next == 1))) begin

        // Pick out the next rotation
        rotate_type <= {my_move_list[current*5+4],my_move_list[current*5+3],my_move_list[
        current*5+2],my_move_list[current*5+1],my_move_list[current*5]};
        start_rotate <= 1;
    end
```

```
        else start_rotate <= 0;

        // When the rotation is complete, update the current pointer to the next rotation and
        the cube configuration
        if (prev_rotate_done != rotate_done && rotate_done == 1) begin
            current <= current + 5'd1;
            cube_config <= new_cube_config;
            trigger_reg <= 1;
        end
        else trigger_reg <= 0;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Locates a cubelet with specific face colors in the current
// cube configuration.  Returns the position relative to an orientation specified
// by the center cubelet colors on each face.
//////////////////////////////////////////////////////////////////////////////////

module Find_Cubelet(
    input clk,                          // clock
    input start_find,                   // goes high when a cubelet should be located
    input [0:8] cubelet_colors,         // lists three cubelet face colors in sequence
    input [0:161] cube_config,          // current cube configuration
    input [0:17] orientation_colors,    // colors of center cubelets in the order up, down,
    front, back, right, left
    output reg [0:8] face_positions,    // lists the faces of the three cubelet faces for the
    given orientation
    output reg find_done,               // goes high when the find operation is done
    output fail                         // goes high if the cubelet is not located
    );

    // Set local parameters that define basic orientations
    localparam N = 3'd0;        // no orientation
    localparam U_basic = 3'd1;  // up
    localparam D_basic = 3'd2;  // down
    localparam F_basic = 3'd3;  // front
    localparam B_basic = 3'd4;  // back
    localparam R_basic = 3'd5;  // right
    localparam L_basic = 3'd6;  // left

    // Allocate registers for storing the position of the up, down, front,
    // back, right, left faces in the cube_config relative to the desired orientation
    reg [0:2] U,D,F,B,R,L;

    // Collect all the edge cubelet colors from the current configuration
    // Each wire is of the form EXY where X and Y represent face positions
    wire [0:5] EUF = {cube_config[21:23],cube_config[57:59]};
    wire [0:5] EFU = {cube_config[57:59],cube_config[21:23]};
    wire [0:5] EUB = {cube_config[3:5],cube_config[156:158]};
    wire [0:5] EBU = {cube_config[156:158],cube_config[3:5]};
    wire [0:5] EUR = {cube_config[15:17],cube_config[84:86]};
    wire [0:5] ERU = {cube_config[84:86],cube_config[15:17]};
    wire [0:5] EUL = {cube_config[9:11],cube_config[30:32]};
    wire [0:5] ELU = {cube_config[30:32],cube_config[9:11]};
    wire [0:5] EDF = {cube_config[111:113],cube_config[75:77]};
    wire [0:5] EFD = {cube_config[75:77],cube_config[111:113]};
    wire [0:5] EDB = {cube_config[129:131],cube_config[138:140]};
    wire [0:5] EBD = {cube_config[138:140],cube_config[129:131]};
    wire [0:5] EDR = {cube_config[123:125],cube_config[102:104]};
    wire [0:5] ERD = {cube_config[102:104],cube_config[123:125]};
    wire [0:5] EDL = {cube_config[117:119],cube_config[48:50]};
    wire [0:5] ELD = {cube_config[48:50],cube_config[117:119]};
    wire [0:5] EFR = {cube_config[69:71],cube_config[90:92]};
    wire [0:5] ERF = {cube_config[90:92],cube_config[69:71]};
    wire [0:5] EFL = {cube_config[63:65],cube_config[42:44]};
    wire [0:5] ELF = {cube_config[42:44],cube_config[63:65]};
    wire [0:5] EBR = {cube_config[150:152],cube_config[96:98]};
    wire [0:5] ERB = {cube_config[96:98],cube_config[150:152]};
    wire [0:5] EBL = {cube_config[144:146],cube_config[36:38]};
    wire [0:5] ELB = {cube_config[36:38],cube_config[144:146]};

    // Collect all the corner cubelet colors from the current configuration
    // Each wire is of the form CXYZ where X, Y, and Z represent face positions
    wire [0:8] CUFL = {cube_config[18:20],cube_config[54:56],cube_config[33:35]};
    wire [0:8] CULF = {cube_config[18:20],cube_config[33:35],cube_config[54:56]};
    wire [0:8] CFUL = {cube_config[54:56],cube_config[18:20],cube_config[33:35]};
    wire [0:8] CFLU = {cube_config[54:56],cube_config[33:35],cube_config[18:20]};
    wire [0:8] CLUF = {cube_config[33:35],cube_config[18:20],cube_config[54:56]};
    wire [0:8] CLFU = {cube_config[33:35],cube_config[54:56],cube_config[18:20]};

    wire [0:8] CUFR = {cube_config[24:26],cube_config[60:62],cube_config[81:83]};
    wire [0:8] CURF = {cube_config[24:26],cube_config[81:83],cube_config[60:62]};
    wire [0:8] CFUR = {cube_config[60:62],cube_config[24:26],cube_config[81:83]};
    wire [0:8] CFRU = {cube_config[60:62],cube_config[81:83],cube_config[24:26]};
    wire [0:8] CRUF = {cube_config[81:83],cube_config[24:26],cube_config[60:62]};
    wire [0:8] CRFU = {cube_config[81:83],cube_config[60:62],cube_config[24:26]};

    wire [0:8] CUBR = {cube_config[6:8],cube_config[159:161],cube_config[87:89]};
    wire [0:8] CURB = {cube_config[6:8],cube_config[87:89],cube_config[159:161]};
    wire [0:8] CBUR = {cube_config[159:161],cube_config[6:8],cube_config[87:89]};
    wire [0:8] CBRU = {cube_config[159:161],cube_config[87:89],cube_config[6:8]};
    wire [0:8] CRUB = {cube_config[87:89],cube_config[6:8],cube_config[159:161]};
    wire [0:8] CRBU = {cube_config[87:89],cube_config[159:161],cube_config[6:8]};

    wire [0:8] CUBL = {cube_config[0:2],cube_config[153:155],cube_config[27:29]};
    wire [0:8] CULB = {cube_config[0:2],cube_config[27:29],cube_config[153:155]};
    wire [0:8] CBUL = {cube_config[153:155],cube_config[0:2],cube_config[27:29]};
    wire [0:8] CBLU = {cube_config[153:155],cube_config[27:29],cube_config[0:2]};
    wire [0:8] CLUB = {cube_config[27:29],cube_config[0:2],cube_config[153:155]};
    wire [0:8] CLBU = {cube_config[27:29],cube_config[153:155],cube_config[0:2]};

    wire [0:8] CDBL = {cube_config[126:128],cube_config[135:137],cube_config[45:47]};
    wire [0:8] CDLB = {cube_config[126:128],cube_config[45:47],cube_config[135:137]};
    wire [0:8] CBDL = {cube_config[135:137],cube_config[126:128],cube_config[45:47]};
    wire [0:8] CBLD = {cube_config[135:137],cube_config[45:47],cube_config[126:128]};
    wire [0:8] CLDB = {cube_config[45:47],cube_config[126:128],cube_config[135:137]};
    wire [0:8] CLBD = {cube_config[45:47],cube_config[135:137],cube_config[126:128]};

    wire [0:8] CDBR = {cube_config[132:134],cube_config[141:143],cube_config[105:107]};
    wire [0:8] CDRB = {cube_config[132:134],cube_config[105:107],cube_config[141:143]};
    wire [0:8] CBDR = {cube_config[141:143],cube_config[132:134],cube_config[105:107]};
    wire [0:8] CBRD = {cube_config[141:143],cube_config[105:107],cube_config[132:134]};
    wire [0:8] CRDB = {cube_config[105:107],cube_config[132:134],cube_config[141:143]};
    wire [0:8] CRBD = {cube_config[105:107],cube_config[141:143],cube_config[132:134]};

    wire [0:8] CDFR = {cube_config[114:116],cube_config[78:80],cube_config[99:101]};
    wire [0:8] CDRF = {cube_config[114:116],cube_config[99:101],cube_config[78:80]};
    wire [0:8] CFDR = {cube_config[78:80],cube_config[114:116],cube_config[99:101]};
    wire [0:8] CFRD = {cube_config[78:80],cube_config[99:101],cube_config[114:116]};
    wire [0:8] CRDF = {cube_config[99:101],cube_config[114:116],cube_config[78:80]};
    wire [0:8] CRFD = {cube_config[99:101],cube_config[78:80],cube_config[114:116]};

    wire [0:8] CDFL = {cube_config[108:110],cube_config[72:74],cube_config[51:53]};
    wire [0:8] CDLF = {cube_config[108:110],cube_config[51:53],cube_config[72:74]};
    wire [0:8] CFDL = {cube_config[72:74],cube_config[108:110],cube_config[51:53]};
    wire [0:8] CFLD = {cube_config[72:74],cube_config[51:53],cube_config[108:110]};
    wire [0:8] CLDF = {cube_config[51:53],cube_config[108:110],cube_config[72:74]};
    wire [0:8] CLFD = {cube_config[51:53],cube_config[72:74],cube_config[108:110]};

    reg intermediate;       // Track whether the orientation has been processed
    reg prev_start;     // Track whether or not start changes

    always @(posedge clk) begin
        prev_start <= start_find;   // Store the status of the start signal

        // If the orientation has already been processed, the find operation will be
        // completed by the next clock cycle
        if (intermediate == 1) begin
            intermediate <= 0;
            find_done <= 1;

            // If there is no third cubelet face color, the cubelet is an edge
            if (cubelet_colors[6:8] == 3'b000) begin
                // Determine the location of the edge cubelet according to the desired
                orientation,
                // leaving the location of the third cubelet face color as no orientation (N)
                case (cubelet_colors[0:5])
                    EUF: face_positions <= {U,F,N};
                    EFU: face_positions <= {F,U,N};
                    EUB: face_positions <= {U,B,N};
                    EBU: face_positions <= {B,U,N};
                    EUR: face_positions <= {U,R,N};
                    ERU: face_positions <= {R,U,N};
                    EUL: face_positions <= {U,L,N};
                    ELU: face_positions <= {L,U,N};
                    EDF: face_positions <= {D,F,N};
                    EFD: face_positions <= {F,D,N};
                    EDB: face_positions <= {D,B,N};
                    EBD: face_positions <= {B,D,N};
                    EDR: face_positions <= {D,R,N};
                    ERD: face_positions <= {R,D,N};
                    EDL: face_positions <= {D,L,N};
                    ELD: face_positions <= {L,D,N};
                    EFR: face_positions <= {F,R,N};
                    ERF: face_positions <= {R,F,N};
                    EFL: face_positions <= {F,L,N};
                    ELF: face_positions <= {L,F,N};
                    EBR: face_positions <= {B,R,N};
                    ERB: face_positions <= {R,B,N};
                    EBL: face_positions <= {B,L,N};
                    ELB: face_positions <= {L,B,N};
                    default: face_positions <= {N,N,N};
                endcase
            end
            // If there is a third cubelet face color, the cubelet is a corner
            else begin
                // Determine the location of the corner cubelet according to the desired
                orientation
                case (cubelet_colors)
                    CUFL: face_positions <= {U,F,L};
                    CULF: face_positions <= {U,L,F};
                    CFUL: face_positions <= {F,U,L};
                    CFLU: face_positions <= {F,L,U};
                    CLUF: face_positions <= {L,U,F};
                    CLFU: face_positions <= {L,F,U};

                    CUFR: face_positions <= {U,F,R};
                    CURF: face_positions <= {U,R,F};
                    CFUR: face_positions <= {F,U,R};
                    CFRU: face_positions <= {F,R,U};
                    CRUF: face_positions <= {R,U,F};
                    CRFU: face_positions <= {R,F,U};

                    CUBR: face_positions <= {U,B,R};
                    CURB: face_positions <= {U,R,B};
                    CBUR: face_positions <= {B,U,R};
                    CBRU: face_positions <= {B,R,U};
                    CRUB: face_positions <= {R,U,B};
                    CRBU: face_positions <= {R,B,U};

                    CUBL: face_positions <= {U,B,L};
                    CULB: face_positions <= {U,L,B};
                    CBUL: face_positions <= {B,U,L};
                    CBLU: face_positions <= {B,L,U};
                    CLUB: face_positions <= {L,U,B};
                    CLBU: face_positions <= {L,B,U};

                    CDBL: face_positions <= {D,B,L};
                    CDLB: face_positions <= {D,L,B};
                    CBDL: face_positions <= {B,D,L};
                    CBLD: face_positions <= {B,L,D};
                    CLDB: face_positions <= {L,D,B};
                    CLBD: face_positions <= {L,B,D};

                    CDBR: face_positions <= {D,B,R};
                    CDRB: face_positions <= {D,R,B};
                    CBDR: face_positions <= {B,D,R};
                    CBRD: face_positions <= {B,R,D};
                    CRDB: face_positions <= {R,D,B};
                    CRBD: face_positions <= {R,B,D};

                    CDFR: face_positions <= {D,F,R};
                    CDRF: face_positions <= {D,R,F};
                    CFDR: face_positions <= {F,D,R};
                    CFRD: face_positions <= {F,R,D};
                    CRDF: face_positions <= {R,D,F};
                    CRFD: face_positions <= {R,F,D};

                    CDFL: face_positions <= {D,F,L};
                    CDLF: face_positions <= {D,L,F};
                    CFDL: face_positions <= {F,D,L};
                    CFLD: face_positions <= {F,L,D};
                    CLDF: face_positions <= {L,D,F};
                    CLFD: face_positions <= {L,F,D};

                    default: face_positions <= {N,N,N};
                endcase
            end
        end

        // If the start signal has just gone high, determine the correspondence between
        // the face positions in the animation (and the cube_config) and the desired cube
        // orientation
        if (start_find != prev_start && start_find == 1) begin
            // Indicate that the orientation will be computed by the next clock cycle
            intermediate <= 1;
            find_done <= 0;

            // Set desired orientation of the up face from cube_config
            case(cube_config[12:14])
                orientation_colors[0:2]: U <= U_basic;
                orientation_colors[3:5]: U <= D_basic;
                orientation_colors[6:8]: U <= F_basic;
                orientation_colors[9:11]: U <= B_basic;
                orientation_colors[12:14]: U <= R_basic;
                orientation_colors[15:17]: U <= L_basic;
                default: U <= N;
            endcase

            // Set desired orientation of the down face from cube_config
            case(cube_config[120:122])
                orientation_colors[0:2]: D <= U_basic;
                orientation_colors[3:5]: D <= D_basic;
                orientation_colors[6:8]: D <= F_basic;
                orientation_colors[9:11]: D <= B_basic;
                orientation_colors[12:14]: D <= R_basic;
                orientation_colors[15:17]: D <= L_basic;
                default: D <= N;
            endcase

            // Set desired orientation of the front face from cube_config
            case(cube_config[66:68])
                orientation_colors[0:2]: F <= U_basic;
                orientation_colors[3:5]: F <= D_basic;
                orientation_colors[6:8]: F <= F_basic;
```

```
                orientation_colors[9:11]: F <= B_basic;
                orientation_colors[12:14]: F <= R_basic;
                orientation_colors[15:17]: F <= L_basic;
                default: F <= N;
            endcase

            // Set desired orientation of the back face from cube_config
            case(cube_config[147:149])
                orientation_colors[0:2]: B <= U_basic;
                orientation_colors[3:5]: B <= D_basic;
                orientation_colors[6:8]: B <= F_basic;
                orientation_colors[9:11]: B <= B_basic;
                orientation_colors[12:14]: B <= R_basic;
                orientation_colors[15:17]: B <= L_basic;
                default: B <= N;
            endcase

            // Set desired orientation of the right face from cube_config
            case(cube_config[93:95])
                orientation_colors[0:2]: R <= U_basic;
                orientation_colors[3:5]: R <= D_basic;
                orientation_colors[6:8]: R <= F_basic;
                orientation_colors[9:11]: R <= B_basic;
                orientation_colors[12:14]: R <= R_basic;
                orientation_colors[15:17]: R <= L_basic;
                default: R <= N;
            endcase

            // Set desired orientation of the left face from cube_config
            case(cube_config[39:41])
                orientation_colors[0:2]: L <= U_basic;
                orientation_colors[3:5]: L <= D_basic;
                orientation_colors[6:8]: L <= F_basic;
                orientation_colors[9:11]: L <= B_basic;
                orientation_colors[12:14]: L <= R_basic;
                orientation_colors[15:17]: L <= L_basic;
                default: L <= N;
            endcase
        end
    end

    // The cubelet is not located if the provided orientation is not valid
    // or the cubelet colors provided do not represent an actual cubelet
    assign fail = (U==N || D==N || F==N || B==N || R==N || L==N || face_positions == {N,N,N});
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Performs a rotation by updating the cube configuration.  Determines
// from the rotation provided in terms of the orientation_colors the absolute
// rotation that must occur in the animation and on the cube configuration.
////////////////////////////////////////////////////////////////////////////////////

module Rotate(
    input clk,                              // clock
    input [4:0] rotate_type,                // rotation in terms of the orientation_colors
    orientation
    input [0:161] cc,                       // current cube configuration
    input [0:17] orientation_colors,        // colors of center cubelets in the order up, down,
    front, back, right, left
    input start_rotate,                     // goes high when rotation should occur
    output reg [0:161] new_cube_config,     // updated cube configuration
    output reg [4:0] animation_type,        // absolute rotation that should occur in the
    animation
    output reg rotate_done                  // goes high when the rotation is complete
    );

    // Define local parameters for each of the 18 rotation types
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    // Get the central colors of each of the faces in the cube configuration in
    // order to determine the correspondences between the animation orientation
    // and the orientation in which the input rotation is defined
    wire [0:2] FaceU = cc[12:14];
    wire [0:2] FaceD = cc[120:122];
    wire [0:2] FaceF = cc[66:68];
    wire [0:2] FaceB = cc[147:149];
    wire [0:2] FaceR = cc[93:95];
    wire [0:2] FaceL = cc[39:41];

    reg intermediate;       // Track whether the orientation has been processed
    reg prev_start;     // Track whether or not start changes

    always @(posedge clk) begin
        prev_start <= start_rotate;     // Store the status of the start signal

        // If the orientation has been processed such that the absolute rotation is known,
        // update the cube configuration
        if (intermediate == 1) begin
            intermediate <= 0;
            rotate_done <= 1;

            // Update the cube configuration according to the absolute rotation
            case (animation_type)
                MRU:
                    new_cube_config <= {cc[0:5],cc[60:62],cc[9:14],cc[69:71],cc[18:23],cc[78:80],
                                                   cc[27:53],
                                                   cc[54:59],cc[114:116],cc[63:68],cc[123:125],cc[72:
                                                   77],cc[132:134],
                                                   cc[99:101],cc[90:92],cc[81:83],cc[102:104],cc[93:95
                                                   ],cc[84:86],cc[105:107],cc[96:98],cc[87:89],
                                                   cc[108:113],cc[141:143],cc[117:122],cc[150:152],cc[
                                                   126:131],cc[159:161],
                                                   cc[135:140],cc[6:8],cc[144:149],cc[15:17],cc[153:
                                                   158],cc[24:26]};
                MRD:
                    new_cube_config <= {cc[0:5],cc[141:143],cc[9:14],cc[150:152],cc[18:23],cc[
                    159:161],
                                                   cc[27:53],
                                                   cc[54:59],cc[6:8],cc[63:68],cc[15:17],cc[72:77],cc[
                                                   24:26],
                                                   cc[87:89],cc[96:98],cc[105:107],cc[84:86],cc[93:95
                                                   ],cc[102:104],cc[81:83],cc[90:92],cc[99:101],
                                                   cc[108:113],cc[60:62],cc[117:122],cc[69:71],cc[126:
                                                   131],cc[78:80],
                                                   cc[135:140],cc[114:116],cc[144:149],cc[123:125],cc[
                                                   153:158],cc[132:134]};
                MFUBD:
                    new_cube_config <= {cc[0:2],cc[57:59],cc[6:11],cc[66:68],cc[15:20],cc[75:77
                    ],cc[24:26],
                                                   cc[27:53],
                                                   cc[54:56],cc[111:113],cc[60:65],cc[120:122],cc[69:
                                                   74],cc[129:131],cc[78:80],
                                                   cc[81:107],
                                                   cc[108:110],cc[138:140],cc[114:119],cc[147:149],cc[
                                                   123:128],cc[156:158],cc[132:134],
                                                   cc[135:137],cc[3:5],cc[141:146],cc[12:14],cc[150:
                                                   155],cc[21:23],cc[159:161]};
                MFDBU:
                    new_cube_config <= {cc[0:2],cc[138:140],cc[6:11],cc[147:149],cc[15:20],cc[
                    156:158],cc[24:26],
                                                   cc[27:53],
                                                   cc[54:56],cc[3:5],cc[60:65],cc[12:14],cc[69:74],cc[
                                                   21:23],cc[78:80],
                                                   cc[81:107],
                                                   cc[108:110],cc[57:59],cc[114:119],cc[66:68],cc[123:
                                                   128],cc[75:77],cc[132:134],
                                                   cc[135:137],cc[111:113],cc[141:146],cc[120:122],cc[
                                                   150:155],cc[129:131],cc[159:161]};
                MLU:
                    new_cube_config <= {cc[54:56],cc[3:8],cc[63:65],cc[12:17],cc[72:74],cc[21:26
                    ],
                                                   cc[33:35],cc[42:44],cc[51:53],cc[30:32],cc[39:41],
                                                   cc[48:50],cc[27:29],cc[36:38],cc[45:47],
                                                   cc[108:110],cc[57:62],cc[117:119],cc[66:71],cc[126:
                                                   128],cc[75:80],
                                                   cc[81:107],
                                                   cc[135:137],cc[111:116],cc[144:146],cc[120:125],cc[
                                                   153:155],cc[129:134],
                                                   cc[0:2],cc[138:143],cc[9:11],cc[147:152],cc[18:20],
                                                   cc[156:161]};
                MLD:
                    new_cube_config <= {cc[135:137],cc[3:8],cc[144:146],cc[12:17],cc[153:155],cc
                    [21:26],
                                                   cc[45:47],cc[36:38],cc[27:29],cc[48:50],cc[39:41],
                                                   cc[30:32],cc[51:53],cc[42:44],cc[33:35],
                                                   cc[0:2],cc[57:62],cc[9:11],cc[66:71],cc[18:20],cc[
                                                   75:80],
                                                   cc[81:107],
                                                   cc[54:56],cc[111:116],cc[63:65],cc[120:125],cc[72:
                                                   74],cc[129:134],
                                                   cc[108:110],cc[138:143],cc[117:119],cc[147:152],cc[
                                                   126:128],cc[156:161]};
                MFR:
                    new_cube_config <= {cc[0:17],cc[51:53],cc[42:44],cc[33:35],
                                                   cc[27:32],cc[108:110],cc[36:41],cc[111:113],cc[45:
                                                   50],cc[114:116],
                                                   cc[72:74],cc[63:65],cc[54:56],cc[75:77],cc[66:68],
                                                   cc[57:59],cc[78:80],cc[69:71],cc[60:62],
                                                   cc[18:20],cc[84:89],cc[21:23],cc[93:98],cc[24:26],
                                                   cc[102:107],
                                                   cc[99:101],cc[90:92],cc[81:83],cc[117:134],
                                                   cc[135:161]};
                MFL:
                    new_cube_config <= {cc[0:17],cc[81:83],cc[90:92],cc[99:101],
                                                   cc[27:32],cc[24:26],cc[36:41],cc[21:23],cc[45:50],
                                                   cc[18:20],
                                                   cc[60:62],cc[69:71],cc[78:80],cc[57:59],cc[66:68],
                                                   cc[75:77],cc[54:56],cc[63:65],cc[72:74],
                                                   cc[114:116],cc[84:89],cc[111:113],cc[93:98],cc[108:
                                                   110],cc[102:107],
                                                   cc[33:35],cc[42:44],cc[51:53],cc[117:134],
                                                   cc[135:161]};
                MURDL:
                    new_cube_config <= {cc[0:8],cc[48:50],cc[39:41],cc[30:32],cc[18:26],
                                                   cc[27:29],cc[117:119],cc[33:38],cc[120:122],cc[42:
                                                   47],cc[123:125],cc[51:53],
                                                   cc[54:80],
                                                   cc[81:83],cc[9:11],cc[87:92],cc[12:14],cc[96:101],
                                                   cc[15:17],cc[105:107],
                                                   cc[108:116],cc[102:104],cc[93:95],cc[84:86],cc[126:
                                                   134],
                                                   cc[135:161]};
                MULDR:
                    new_cube_config <= {cc[0:8],cc[84:86],cc[93:95],cc[102:104],cc[18:26],
                                                   cc[27:29],cc[15:17],cc[33:38],cc[12:14],cc[42:47],
                                                   cc[9:11],cc[51:53],
                                                   cc[54:80],
                                                   cc[81:83],cc[123:125],cc[87:92],cc[120:122],cc[96:
                                                   101],cc[117:119],cc[105:107],
                                                   cc[108:116],cc[30:32],cc[39:41],cc[48:50],cc[126:
                                                   134],
                                                   cc[135:161]};
                MBR:
                    new_cube_config <= {cc[45:47],cc[36:38],cc[27:29],cc[9:26],
                                                   cc[126:128],cc[30:35],cc[129:131],cc[39:44],cc[132:
                                                   134],cc[48:53],
                                                   cc[54:80],
                                                   cc[81:86],cc[0:2],cc[90:95],cc[3:5],cc[99:104],cc[6
                                                   :8],
                                                   cc[108:125],cc[105:107],cc[96:98],cc[87:89],
                                                   cc[141:143],cc[150:152],cc[159:161],cc[138:140],cc[
                                                   147:149],cc[156:158],cc[135:137],cc[144:146],cc[153
                                                   :155]};
                MBL:
                    new_cube_config <= {cc[87:89],cc[96:98],cc[105:107],cc[9:26],
                                                   cc[6:8],cc[30:35],cc[3:5],cc[39:44],cc[0:2],cc[48:
                                                   53],
                                                   cc[54:80],
                                                   cc[81:86],cc[132:134],cc[90:95],cc[129:131],cc[99:
                                                   104],cc[126:128],
                                                   cc[108:125],cc[27:29],cc[36:38],cc[45:47],
                                                   cc[153:155],cc[144:146],cc[135:137],cc[156:158],cc[
                                                   147:149],cc[138:140],cc[159:161],cc[150:152],cc[141
                                                   :143]};
                MUR:
                    new_cube_config <= {cc[6:8],cc[15:17],cc[24:26],cc[3:5],cc[12:14],cc[21:23],
                    cc[0:2],cc[9:11],cc[18:20],
                                                   cc[159:161],cc[156:158],cc[153:155],cc[36:53],
                                                   cc[27:35],cc[63:80],
                                                   cc[54:62],cc[90:107],
                                                   cc[108:134],
                                                   cc[135:152],cc[87:89],cc[84:86],cc[81:83]};
                MUL:
                    new_cube_config <= {cc[18:20],cc[9:11],cc[0:2],cc[21:23],cc[12:14],cc[3:5],
                    cc[24:26],cc[15:17],cc[6:8],
                                                   cc[54:62],cc[36:53],
                                                   cc[81:89],cc[63:80],
                                                   cc[159:161],cc[156:158],cc[153:155],cc[90:107],
                                                   cc[108:134],
                                                   cc[135:152],cc[33:35],cc[30:32],cc[27:29]};
                MFRBL:
                    new_cube_config <= {cc[0:26],
                                                   cc[27:35],cc[150:152],cc[147:149],cc[144:146],cc[45
                                                   :53],
                                                   cc[54:62],cc[36:44],cc[72:80],
                                                   cc[81:89],cc[63:71],cc[99:107],
                                                   cc[108:134],
                                                   cc[135:143],cc[96:98],cc[93:95],cc[90:92],cc[153:
                                                   161]};
                MFLBR:
                    new_cube_config <= {cc[0:26],
                                                   cc[27:35],cc[63:71],cc[45:53],
                                                   cc[54:62],cc[90:98],cc[72:80],
                                                   cc[81:89],cc[150:152],cc[147:149],cc[144:146],cc[99
                                                   :107],
                                                   cc[108:134],
                                                   cc[135:143],cc[42:44],cc[39:41],cc[36:38],cc[153:
                                                   161]};
                MDR:
                    new_cube_config <= {cc[0:26],
                                                   cc[27:44],cc[141:143],cc[138:140],cc[135:137],
                                                   cc[54:71],cc[45:53],
                                                   cc[81:98],cc[72:80],
                                                   cc[126:128],cc[117:119],cc[108:110],cc[129:131],cc[
                                                   120:122],cc[111:113],cc[132:134],cc[123:125],cc[114
                                                   :116],
                                                   cc[105:107],cc[102:104],cc[99:101],cc[144:161]};
                MDL:
                    new_cube_config <= {cc[0:26],
                                                   cc[27:44],cc[72:80],
                                                   cc[54:71],cc[99:107],
                                                   cc[81:98],cc[141:143],cc[138:140],cc[135:137],
                                                   cc[114:116],cc[123:125],cc[132:134],cc[111:113],cc[
                                                   120:122],cc[129:131],cc[108:110],cc[117:119],cc[126
                                                   :128],
                                                   cc[51:53],cc[48:50],cc[45:47],cc[144:161]};
                  endcase
              end

              // If the start signal has just gone high, determine the absolute rotation
              if (start_rotate != prev_start && start_rotate == 1) begin
                  // Indicate that the absolute rotation will be know by the next clock cycle
                  intermediate <= 1;
                  rotate_done <= 0;

                  // Determine the absolute rotation that corresponds to the provided rotation in
                  // terms of orientation_colors
                  // Knowledge of the location in the cube_configuration of one of face colors
                  // defined in orientation_colors is sufficient
                  case (rotate_type)
                MRU:
                    // Compute the absolute rotation corresponding to MRU from the defined
                    right face color
                    case (orientation_colors[12:14])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MRD:
                    // Compute the absolute rotation corresponding to MRD from the defined
                    right face color
                    case (orientation_colors[12:14])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
                MFUBD:
                    // Compute the absolute rotation corresponding to MFUBD from the defined
                    right face color
                    case (orientation_colors[12:14])
                        FaceU: animation_type <= MFLBR;
                        FaceD: animation_type <= MFRBL;
                        FaceF: animation_type <= MURDL;
                        FaceB: animation_type <= MULDR;
                        FaceR: animation_type <= MFUBD;
                        FaceL: animation_type <= MFDBU;
                    endcase
                MFDBU:
                    // Compute the absolute rotation corresponding to MFDBU from the defined
                    right face color
                    case (orientation_colors[12:14])
                        FaceU: animation_type <= MFRBL;
                        FaceD: animation_type <= MFLBR;
                        FaceF: animation_type <= MULDR;
                        FaceB: animation_type <= MURDL;
                        FaceR: animation_type <= MFDBU;
                        FaceL: animation_type <= MFUBD;
                    endcase
                MLU:
                    // Compute the absolute rotation corresponding to MLU from the defined left
                    face color
                    case (orientation_colors[15:17])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
                MLD:
                    // Compute the absolute rotation corresponding to MLD from the defined left
                    face color
                    case (orientation_colors[15:17])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MFR:
                    // Compute the absolute rotation corresponding to MFR from the defined
                    front face color
                    case (orientation_colors[6:8])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MFL:
                    // Compute the absolute rotation corresponding to MFL from the defined
                    front face color
                    case (orientation_colors[6:8])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
                MURDL:
                    // Compute the absolute rotation corresponding to MURDL from the defined
                    front face color
                    case (orientation_colors[6:8])
                        FaceU: animation_type <= MFLBR;
                        FaceD: animation_type <= MFRBL;
                        FaceF: animation_type <= MURDL;
                        FaceB: animation_type <= MULDR;
                        FaceR: animation_type <= MFUBD;
                        FaceL: animation_type <= MFDBU;
                    endcase
                MULDR:
                    // Compute the absolute rotation corresponding to MULDR from the defined
                    front face color
                    case (orientation_colors[6:8])
                        FaceU: animation_type <= MFRBL;
                        FaceD: animation_type <= MFLBR;
                        FaceF: animation_type <= MULDR;
                        FaceB: animation_type <= MURDL;
                        FaceR: animation_type <= MFDBU;
                        FaceL: animation_type <= MFUBD;
                    endcase
                MBR:
                    // Compute the absolute rotation corresponding to MBR from the defined back
                    face color
                    case (orientation_colors[9:11])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
                MBL:
                    // Compute the absolute rotation corresponding to MBL from the defined back
                    face color
                    case (orientation_colors[9:11])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MUR:
                    // Compute the absolute rotation corresponding to MUR from the defined up
                    face color
                    case (orientation_colors[0:2])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
                MUL:
                    // Compute the absolute rotation corresponding to MUL from the defined up
                    face color
                    case (orientation_colors[0:2])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MFRBL:
                    // Compute the absolute rotation corresponding to MFRBL from the defined up
                    face color
                    case (orientation_colors[0:2])
                        FaceU: animation_type <= MFRBL;
                        FaceD: animation_type <= MFLBR;
                        FaceF: animation_type <= MULDR;
                        FaceB: animation_type <= MURDL;
                        FaceR: animation_type <= MFDBU;
                        FaceL: animation_type <= MFUBD;
                    endcase
                MFLBR:
                    // Compute the absolute rotation corresponding to MFLBR from the defined up
                    face color
                    case (orientation_colors[0:2])
                        FaceU: animation_type <= MFLBR;
                        FaceD: animation_type <= MFRBL;
                        FaceF: animation_type <= MURDL;
                        FaceB: animation_type <= MULDR;
                        FaceR: animation_type <= MFUBD;
                        FaceL: animation_type <= MFDBU;
                    endcase
                MDR:
                    // Compute the absolute rotation corresponding to MDR from the defined down
                    face color
                    case (orientation_colors[3:5])
                        FaceU: animation_type <= MUL;
                        FaceD: animation_type <= MDR;
                        FaceF: animation_type <= MFR;
                        FaceB: animation_type <= MBL;
                        FaceR: animation_type <= MRU;
                        FaceL: animation_type <= MLD;
                    endcase
                MDL:
                    // Compute the absolute rotation corresponding to MDL from the defined down
                    face color
                    case (orientation_colors[3:5])
                        FaceU: animation_type <= MUR;
                        FaceD: animation_type <= MDL;
                        FaceF: animation_type <= MFL;
                        FaceB: animation_type <= MBR;
                        FaceR: animation_type <= MRD;
                        FaceL: animation_type <= MLU;
                    endcase
            endcase
        end
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Sends the cube configuration and animation type from the first
// labkit to the second labkit in serial fashion
//////////////////////////////////////////////////////////////////////////////////

module Serial_send(
    input clock,                        // clock
    input [0:161] cube_config,          // cube configuration to send
    input [4:0] animation_type,         // animation to send
    input trigger,                      // goes high when data should be sent
    output clock_pulse,                 // square clock wave for sending data
    output reg sync_pulse,              // sync pulse for sending data
    output reg data,                    // serial data line
    output state_wire                   // state of sending serial communication
    );

    // Send the cube configuration first, followed by the animation type
    wire [0:166] combined_data;
    assign combined_data = {cube_config, animation_type};

    reg [7:0] counter = 8'd0;   // counter for creating a slow clock pulse
    reg [7:0] position = 8'd0;  // index of data being sent
    reg [1:0] state = 2'd0;     // state of sending serial communication
    reg clock_pulse_reg = 0;    // reg for square clock wave
    reg prev_trigger;           // keep the previous trigger to know when trigger goes high

    assign state_wire = state;
    assign clock_pulse = clock_pulse_reg;

    localparam waiting_for_trigger = 2'd0;  // state where no data being sent
    localparam do_sync = 2'd1;              // state for sending sync pulse
    localparam send_data = 2'd2;            // state for sending data

    always @(posedge clock) begin
        prev_trigger <= trigger;
        counter <= counter + 1;

        // Enter the state to send the sync pulse if the trigger has gone high
        if (state == waiting_for_trigger && trigger != prev_trigger && trigger) begin
            state <= do_sync;
        end

        if (counter == 8'd255) begin
            // Alternate the clock signal every 256 counts
            clock_pulse_reg <= ~clock_pulse_reg;
            if (clock_pulse_reg == 0) begin
                case (state)
                    // Create a sync pulse for one entire clock signal period
                    do_sync: begin
                        sync_pulse <= 1;
                        position <= 8'd0;
```

```
                          state <= send_data;
                      end
                      // Send the combined_data one bit at a time
                      send_data: begin
                          sync_pulse <= 0;
                          position <= position + 1;
                          data <= combined_data[position];
                          // Once all data has been transmitted, return to the waiting state
                          if (position == 8'd166) begin
                              state <= waiting_for_trigger;
                          end
                      end
                  endcase
              end
          end
      end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the position and
// orientation of an edge cubelet on the top face
//////////////////////////////////////////////////////////////////////////////////

module Top_Face_Edge(
    input [8:0] face_positions,          // position of the edge cubelet
    output reg [199:0] move_list,        // rotation sequence to be performed
    output reg [5:0] num_moves,          // number of moves in the rotation sequence
    output reg fail                      // goes high if cube configuration is not valid to start
    );

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    always @(*) begin
        // Based on the location of the edge cubelet, determine the appropriate sequence of moves
        // The fail flag is set if the location of the edge cubelet is invalid
        case (face_positions)
            {U,F,N}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {U,R,N}: begin move_list = {MFRBL,MRU,MFLBR,MRD}; num_moves = 6'd4; fail = 0; end
            {U,B,N}: begin move_list = {MUR,MLU,MUR,MUR,MLD,MUR}; num_moves = 6'd6; fail = 0; end
            {U,L,N}: begin move_list = {MFLBR,MLU,MFRBL,MLD}; num_moves = 6'd4; fail = 0; end
            {F,D,N}: begin move_list = {MFRBL,MRU,MFLBR,MFL}; num_moves = 6'd4; fail = 0; end
            {F,R,N}: begin move_list = {MFRBL,MRU,MFLBR}; num_moves = 6'd3; fail = 0; end
            {F,U,N}: begin move_list = {MFRBL,MRU,MFLBR,MFR}; num_moves = 6'd4; fail = 0; end
            {F,L,N}: begin move_list = {MFLBR,MLU,MFRBL}; num_moves = 6'd3; fail = 0; end
            {D,B,N}: begin move_list = {MFR,MFR,MDR,MDR}; num_moves = 6'd4; fail = 0; end
            {D,R,N}: begin move_list = {MFR,MFR,MDL}; num_moves = 6'd3; fail = 0; end
            {D,F,N}: begin move_list = {MFR,MFR}; num_moves = 6'd2; fail = 0; end
            {D,L,N}: begin move_list = {MFR,MFR,MDR}; num_moves = 6'd3; fail = 0; end
            {R,F,N}: begin move_list = {MFL}; num_moves = 6'd1; fail = 0; end
            {R,D,N}: begin move_list = {MFL,MUL,MRU,MUR}; num_moves = 6'd4; fail = 0; end
            {R,B,N}: begin move_list = {MFL,MUL,MRU,MRU,MUR}; num_moves = 6'd5; fail = 0; end
            {R,U,N}: begin move_list = {MFL,MRD}; num_moves = 6'd2; fail = 0; end
            {L,F,N}: begin move_list = {MFR}; num_moves = 6'd1; fail = 0; end
            {L,U,N}: begin move_list = {MFR,MLD}; num_moves = 6'd2; fail = 0; end
            {L,B,N}: begin move_list = {MFR,MUR,MLU,MLU,MUL}; num_moves = 6'd5; fail = 0; end
            {L,D,N}: begin move_list = {MFR,MUR,MLU,MUL}; num_moves = 6'd4; fail = 0; end
            {B,U,N}: begin move_list = {MFL,MFLBR,MRD,MFRBL}; num_moves = 6'd4; fail = 0; end
            {B,R,N}: begin move_list = {MUL,MRD,MUR}; num_moves = 6'd3; fail = 0; end
            {B,D,N}: begin move_list = {MFL,MUL,MRU,MUR,MDL}; num_moves = 6'd5; fail = 0; end
            {B,L,N}: begin move_list = {MUR,MLD,MUL}; num_moves = 6'd3; fail = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the position and
// orientation of a corner cubelet on the top face
//////////////////////////////////////////////////////////////////////////////////

module Top_Face_Corner(
    input [8:0] face_positions,          // position of the corner cubelet
    output reg [199:0] move_list,        // rotation sequence to be performed
    output reg [5:0] num_moves,          // number of moves in the rotation sequence
    output reg fail                      // goes high if cube configuration is not valid to start
    );

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    always @(*) begin
        // Based on the location of the corner cubelet, determine the appropriate sequence of
        moves
        // The fail flag is set if the location of the corner cubelet is invalid
        case (face_positions)
            {U,F,L},{U,L,F}: begin move_list = {MRU,MDL,MRD,MLU,MDR,MLD}; num_moves = 6'd6; fail
             = 0; end
            {F,U,L},{F,L,U}: begin move_list = {MFL,MDL,MFR,MFR,MDR,MDR,MFL}; num_moves = 6'd7;
            fail = 0; end
            {L,U,F},{L,F,U}: begin move_list = {MFDBU,MDR,MFUBD}; num_moves = 6'd3; fail = 0; end
            {U,F,R},{U,R,F}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {F,U,R},{F,R,U}: begin move_list = {MRU,MDR,MRD,MDL,MRU,MDR,MRD}; num_moves = 6'd7;
            fail = 0; end
            {R,U,F},{R,F,U}: begin move_list = {MRU,MDL,MRD,MDR,MRU,MDL,MRD}; num_moves = 6'd7;
            fail = 0; end

            {U,B,R},{U,R,B}: begin move_list = {MRU,MDL,MRD,MDR,MRD,MDL,MDL,MRU}; num_moves =
            6'd8; fail = 0; end
            {B,U,R},{B,R,U}: begin move_list = {MFL,MDL,MFR,MRD,MDL,MRU}; num_moves = 6'd6; fail
             = 0; end
            {R,U,B},{R,B,U}: begin move_list = {MRU,MDR,MRD,MRD,MDL,MDL,MRU}; num_moves = 6'd7;
            fail = 0; end

            {U,B,L},{U,L,B}: begin move_list = {MRU,MLD,MDR,MDR,MRD,MLU}; num_moves = 6'd6; fail
             = 0; end
            {B,U,L},{B,L,U}: begin move_list = {MRU,MDR,MRD,MDR,MLD,MDR,MLU}; num_moves = 6'd7;
            fail = 0; end
            {L,U,B},{L,B,U}: begin move_list = {MFL,MDL,MFR,MLD,MDL,MLU}; num_moves = 6'd6; fail
             = 0; end

            {D,B,L},{D,L,B}: begin move_list = {MRU,MDR,MRD,MDL,MRU,MDL,MRD}; num_moves = 6'd7;
            fail = 0; end
            {B,D,L},{B,L,D}: begin move_list = {MRU,MDR,MDR,MRD}; num_moves = 6'd4; fail = 0; end
            {L,D,B},{L,B,D}: begin move_list = {MFL,MDR,MDR,MFR}; num_moves = 6'd4; fail = 0; end

            {D,B,R},{D,R,B}: begin move_list = {MFL,MDL,MFR,MDR,MFL,MDL,MDL,MFR}; num_moves =
            6'd8; fail = 0; end
            {B,D,R},{B,R,D}: begin move_list = {MFL,MDL,MFR}; num_moves = 6'd3; fail = 0; end
            {R,D,B},{R,B,D}: begin move_list = {MRU,MDR,MRD,MDL,MDL}; num_moves = 6'd5; fail = 0
            ; end

            {D,F,R},{D,R,F}: begin move_list = {MRU,MDR,MRD,MDL,MRU,MDR,MDR,MRD,MDL}; num_moves
            = 6'd9; fail = 0; end
            {F,D,R},{F,R,D}: begin move_list = {MFL,MDR,MFR}; num_moves = 6'd3; fail = 0; end
            {R,D,F},{R,F,D}: begin move_list = {MRU,MDL,MRD}; num_moves = 6'd3; fail = 0; end

            {D,F,L},{D,L,F}: begin move_list = {MRU,MDR,MRD,MDL,MRU,MDR,MDR,MRD}; num_moves =
            6'd8; fail = 0; end
            {F,D,L},{F,L,D}: begin move_list = {MRU,MDL,MRD,MDR}; num_moves = 6'd4; fail = 0; end
            {L,D,F},{L,F,D}: begin move_list = {MRU,MDR,MRD}; num_moves = 6'd3; fail = 0; end

            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the position and
// orientation of a edge cubelet in the middle layer
////////////////////////////////////////////////////////////////////////////////////

module Middle_Edge(
    input [8:0] face_positions,         // position of the edge cubelet
    output reg [199:0] move_list,       // rotation sequence to be performed
    output reg [5:0] num_moves,         // number of moves in the rotation sequence
    output reg fail                     // goes high if cube configuration is not valid to start
    );

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    always @(*) begin
        // Based on the location of the edge cubelet, determine the appropriate sequence of moves
        // The fail flag is set if the location of the edge cubelet is invalid
        case (face_positions)
            {L,D,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD}; num_moves = 6'd7; fail = 0
            ; end
            {F,D,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDL}; num_moves = 6'd8; fail
             = 0; end
            {R,D,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDL,MDL}; num_moves = 6'd9;
            fail = 0; end
            {B,D,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDR}; num_moves = 6'd8; fail
             = 0; end
            {D,B,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR}; num_moves = 6'd7; fail = 0
            ; end
            {D,R,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR,MDR}; num_moves = 6'd8; fail
             = 0; end
            {D,F,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR,MDR,MDR}; num_moves = 6'd9;
            fail = 0; end
            {D,L,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR,MDL}; num_moves = 6'd8; fail
             = 0; end
            {F,R,N}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {R,F,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDR,MFL,MDL,MFR,MDR,MRU,MDR,
            MRD}; num_moves = 6'd15; fail = 0; end
            {F,L,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDL,MDL,MLU,MDL,MLD,MDR,MFR,
            MDR,MFL}; num_moves = 6'd16; fail = 0; end
            {L,F,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDR,MFR,MDR,MFL,MDL,MLU,MDL,
            MLD}; num_moves = 6'd15; fail = 0; end
            {R,B,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDL,MBL,MDR,MBR,MDL,MRD,MDL,
            MRU}; num_moves = 6'd15; fail = 0; end
            {B,R,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR,MDR,MDR,MBL,MDR,MBR,MDL,MRD,
            MDL,MRU}; num_moves = 6'd16; fail = 0; end
            {B,L,N}: begin move_list = {MRU,MDR,MRD,MDL,MFL,MDL,MFR,MDR,MDR,MBR,MDL,MBL,MDR,MLD,
            MDR,MLU}; num_moves = 6'd16; fail = 0; end
            {L,B,N}: begin move_list = {MFL,MDL,MFR,MDR,MRU,MDR,MRD,MDL,MBR,MDL,MBL,MDR,MLD,MDR,
            MLU}; num_moves = 6'd15; fail = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the position of
// all the corner cubelets on the bottom layer
//////////////////////////////////////////////////////////////////////////////////

module Bottom_Corner_Position(
    input [8:0] RYBposition,                // position of the red-yellow-blue cubelet
    input [8:0] RWBposition,                // position of the red-white-blue cubelet
    input [8:0] OWBposition,                // position of the orange-white-blue cubelet
    input [8:0] OYBposition,                // position of the orange-yellow-blue cubelet
    output reg [199:0] move_list,           // rotation sequence to be performed
    output reg [5:0] num_moves,             // number of moves in the rotation sequence
    output reg fail                         // goes high if cube configuration is not valid to
    start
    );

    // Not all of the position information for the corner cubelets is needed
    reg [1:0] RYBposition_simple;
    reg [1:0] RWBposition_simple;
    reg [1:0] OWBposition_simple;
    reg [1:0] OYBposition_simple;

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;
    // Define simpler locations for the corner cubelets
    localparam BL = 2'd0;   // bottom left
    localparam BR = 2'd1;   // bottom right
    localparam TR = 2'd2;   // top right
    localparam TL = 2'd3;   // top left

    // Define two common rotation subsequences
    localparam A_seq = {MUL,MUL,MLD,MUL,MLU,MFL,MUL,MFR,MLD,MUR,MLU};
    localparam B_seq = {MLD,MUL,MLU,MFL,MUL,MUL,MFR,MLD,MUR,MLU};

    always @(*) begin
        // Determine the simpler position of the red-yellow-blue cubelet
        case(RYBposition)
            {U,L,F},{U,F,L},{L,U,F},{L,F,U},{F,U,L},{F,L,U}: RYBposition_simple = BL;
            {U,R,F},{U,F,R},{R,U,F},{R,F,U},{F,U,R},{F,R,U}: RYBposition_simple = BR;
            {U,R,B},{U,B,R},{R,U,B},{R,B,U},{B,U,R},{B,R,U}: RYBposition_simple = TR;
            {U,L,B},{U,B,L},{L,U,B},{L,B,U},{B,U,L},{B,L,U}: RYBposition_simple = TL;
            default: fail = 1;
        endcase
        // Determine the simpler position of the red-white-blue cubelet
        case (RWBposition)
            {U,L,F},{U,F,L},{L,U,F},{L,F,U},{F,U,L},{F,L,U}: RWBposition_simple = BL;
            {U,R,F},{U,F,R},{R,U,F},{R,F,U},{F,U,R},{F,R,U}: RWBposition_simple = BR;
            {U,R,B},{U,B,R},{R,U,B},{R,B,U},{B,U,R},{B,R,U}: RWBposition_simple = TR;
            {U,L,B},{U,B,L},{L,U,B},{L,B,U},{B,U,L},{B,L,U}: RWBposition_simple = TL;
            default: fail = 1;
        endcase
        // Determine the simpler position of the orange-white-blue cubelet
        case (OWBposition)
            {U,L,F},{U,F,L},{L,U,F},{L,F,U},{F,U,L},{F,L,U}: OWBposition_simple = BL;
            {U,R,F},{U,F,R},{R,U,F},{R,F,U},{F,U,R},{F,R,U}: OWBposition_simple = BR;
            {U,R,B},{U,B,R},{R,U,B},{R,B,U},{B,U,R},{B,R,U}: OWBposition_simple = TR;
            {U,L,B},{U,B,L},{L,U,B},{L,B,U},{B,U,L},{B,L,U}: OWBposition_simple = TL;
            default: fail = 1;
        endcase
        // Determine the simpler position of the orange-yellow-blue cubelet
        case (OYBposition)
            {U,L,F},{U,F,L},{L,U,F},{L,F,U},{F,U,L},{F,L,U}: OYBposition_simple = BL;
            {U,R,F},{U,F,R},{R,U,F},{R,F,U},{F,U,R},{F,R,U}: OYBposition_simple = BR;
            {U,R,B},{U,B,R},{R,U,B},{R,B,U},{B,U,R},{B,R,U}: OYBposition_simple = TR;
            {U,L,B},{U,B,L},{L,U,B},{L,B,U},{B,U,L},{B,L,U}: OYBposition_simple = TL;
            default: fail = 1;
        endcase

        // Based on the simplified locations of the four bottom corner cubelets, determine the
        appropriate sequence of moves
        // The fail flag is set if the starting positions of the corner cubelets are invalid
        case ({RYBposition_simple,RWBposition_simple,OWBposition_simple,OYBposition_simple})
            {BR,BL,TL,TR}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {BR,BL,TR,TL}: begin move_list = {MUR,MUR,A_seq,MUR,MUR}; num_moves = 6'd15; fail =
            0; end
            {BR,TL,BL,TR}: begin move_list = {MUL,A_seq,MUR}; num_moves = 6'd13; fail = 0; end
            {BR,TL,TR,BL}: begin move_list = {MUR,A_seq}; num_moves = 6'd12; fail = 0; end
            {BR,TR,BL,TL}: begin move_list = {A_seq,MUL}; num_moves = 6'd12; fail = 0; end
            {BR,TR,TL,BL}: begin move_list = {B_seq,MUL}; num_moves = 6'd11; fail = 0; end
            {BL,BR,TL,TR}: begin move_list = {A_seq}; num_moves = 6'd11; fail = 0; end
            {BL,BR,TR,TL}: begin move_list = {B_seq}; num_moves = 6'd10; fail = 0; end
            {BL,TL,BR,TR}: begin move_list = {MUR,MUR,A_seq,MUL}; num_moves = 6'd14; fail = 0;
            end
            {BL,TL,TR,BR}: begin move_list = {MUR}; num_moves = 6'd1; fail = 0; end
            {BL,TR,BR,TL}: begin move_list = {MUR,A_seq,MUR}; num_moves = 6'd13; fail = 0; end
            {BL,TR,TL,BR}: begin move_list = {MUL,A_seq,MUR,MUR}; num_moves = 6'd14; fail = 0;
            end
            {TL,BR,BL,TR}: begin move_list = {MUR,A_seq,MUR,MUR}; num_moves = 6'd14; fail = 0;
            end
            {TL,BR,TR,BL}: begin move_list = {MUL,A_seq,MUL}; num_moves = 6'd13; fail = 0; end
            {TL,BL,BR,TR}: begin move_list = {MUL,B_seq}; num_moves = 6'd11; fail = 0; end
            {TL,BL,TR,BR}: begin move_list = {A_seq,MUR}; num_moves = 6'd12; fail = 0; end
            {TL,TR,BR,BL}: begin move_list = {MUR,MUR}; num_moves = 6'd2; fail = 0; end
            {TL,TR,BL,BR}: begin move_list = {MUR,MUR,A_seq}; num_moves = 6'd13; fail = 0; end
            {TR,BR,BL,TL}: begin move_list = {MUL}; num_moves = 6'd1; fail = 0; end
            {TR,BR,TL,BL}: begin move_list = {MUR,MUR,A_seq,MUR}; num_moves = 6'd14; fail = 0;
            end
            {TR,BL,BR,TL}: begin move_list = {MUL,A_seq}; num_moves = 6'd12; fail = 0; end
            {TR,BL,TL,BR}: begin move_list = {MUR,A_seq,MUL}; num_moves = 6'd13; fail = 0; end
            {TR,TL,BR,BL}: begin move_list = {A_seq,MUR,MUR}; num_moves = 6'd13; fail = 0; end
            {TR,TL,BL,BR}: begin move_list = {MUR,MUR,B_seq}; num_moves = 6'd12; fail = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the orientation of
// all the corner cubelets on the bottom layer
//////////////////////////////////////////////////////////////////////////////////

module Bottom_Corner_Orientation(
    input [0:8] BRYposition,            // position of the blue-red-yellow cubelet
    input [0:8] BRWposition,            // position of the blue-red-white cubelet
    input [0:8] BOWposition,            // position of the blue-orange-white cubelet
    input [0:8] BOYposition,            // position of the blue-orange-yellow cubelet
    output reg [199:0] move_list,       // rotation sequence to be performed
    output reg [5:0] num_moves,         // number of moves in the rotation sequence
    output reg fail                     // goes high if cube configuration is not valid to start
    );

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    // Define two common rotation subsequences
    localparam A_seq = {MUL,MUL,MRD,MUL,MUL,MRU,MUL,MRD,MUL,MRU};
    localparam B_seq = {MUR,MUR,MLD,MUR,MUR,MLU,MUR,MLD,MUR,MLU};

    always @(*) begin
        // Based on orientations of the four bottom corner cubelets, determine the appropriate
        sequence of moves
        // The fail flag is set if the starting orientations of the corner cubelets are invalid
        case ({BRYposition[0:2],BRWposition[0:2],BOWposition[0:2],BOYposition[0:2]})
            {U,U,U,U}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {F,U,B,R}: begin move_list = {A_seq}; num_moves = 6'd10; fail = 0; end
            {F,L,U,R}: begin move_list = {MUL,A_seq,MUR}; num_moves = 6'd12; fail = 0; end
            {F,L,B,U}: begin move_list = {MUR,MUR,A_seq,MUL,MUL}; num_moves = 6'd14; fail = 0;
            end
            {U,L,B,R}: begin move_list = {MUR,A_seq,MUL}; num_moves = 6'd12; fail = 0; end
            {U,F,L,B}: begin move_list = {B_seq}; num_moves = 6'd10; fail = 0; end
            {R,U,L,B}: begin move_list = {MUL,B_seq,MUR}; num_moves = 6'd12; fail = 0; end
            {R,F,U,B}: begin move_list = {MUR,MUR,B_seq,MUL,MUL}; num_moves = 6'd14; fail = 0;
            end
            {R,F,L,U}: begin move_list = {MUR,B_seq,MUL}; num_moves = 6'd12; fail = 0; end
            {U,F,B,U}: begin move_list = {MUR,MUR,B_seq,MUR,MUR,A_seq}; num_moves = 6'd24; fail
            = 0; end
            {U,U,L,R}: begin move_list = {MUR,B_seq,MUR,MUR,A_seq,MUR}; num_moves = 6'd24; fail
            = 0; end
            {F,U,U,B}: begin move_list = {B_seq,MUR,MUR,A_seq,MUR,MUR}; num_moves = 6'd24; fail
            = 0; end
            {R,L,U,U}: begin move_list = {MUL,B_seq,MUR,MUR,A_seq,MUL}; num_moves = 6'd24; fail
            = 0; end
            {F,F,U,U}: begin move_list = {B_seq,A_seq}; num_moves = 6'd20; fail = 0; end
            {U,L,L,U}: begin move_list = {MUL,B_seq,A_seq,MUR}; num_moves = 6'd22; fail = 0; end
            {U,U,B,B}: begin move_list = {MUR,MUR,B_seq,A_seq,MUR,MUR}; num_moves = 6'd24; fail
            = 0; end
            {R,U,U,R}: begin move_list = {MUR,B_seq,A_seq,MUL}; num_moves = 6'd22; fail = 0; end
            {U,F,U,R}: begin move_list = {MUR,B_seq,MUL,A_seq}; num_moves = 6'd22; fail = 0; end
            {F,U,L,U}: begin move_list = {B_seq,MUL,A_seq,MUR}; num_moves = 6'd22; fail = 0; end
            {U,L,U,B}: begin move_list = {MUL,B_seq,MUL,A_seq,MUR,MUR}; num_moves = 6'd24; fail
            = 0; end
            {R,U,B,U}: begin move_list = {MUR,MUR,B_seq,MUL,A_seq,MUL}; num_moves = 6'd24; fail
            = 0; end
            {F,L,L,B}: begin move_list = {MUR,A_seq,MUL,A_seq}; num_moves = 6'd22; fail = 0; end
            {R,L,B,B}: begin move_list = {A_seq,MUL,A_seq,MUR}; num_moves = 6'd22; fail = 0; end
            {R,F,B,R}: begin move_list = {MUL,A_seq,MUL,A_seq,MUR,MUR}; num_moves = 6'd24; fail
            = 0; end
            {F,F,L,R}: begin move_list = {MUR,MUR,A_seq,MUL,A_seq,MUL}; num_moves = 6'd24; fail
            = 0; end
            {R,L,L,R}: begin move_list = {MUL,MUL,A_seq,MUR,MUR,A_seq}; num_moves = 6'd24; fail
            = 0; end
            {F,F,B,B}: begin move_list = {MUR,A_seq,MUR,MUR,A_seq,MUR}; num_moves = 6'd24; fail
            = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the position of
// all the edge cubelets on the bottom layer given the position of two edge cubelets
////////////////////////////////////////////////////////////////////////////////

module Bottom_Edge_Position(
    input [0:8] RBposition,            // position of the red-blue cubelet
    input [0:8] WBposition,            // position of the white-blue cubelet
    output reg [199:0] move_list,      // rotation sequence to be performed
    output reg [5:0] num_moves,        // number of moves in the rotation sequence
    output reg fail                    // goes high if cube configuration is not valid to start
    );

    // Not all of the position information for the edge cubelets is needed
    reg [1:0] RBposition_simple;
    reg [1:0] WBposition_simple;

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    // Define simpler locations for the edge cubelets
    localparam top = 2'd0;
    localparam bot = 2'd1;
    localparam left = 2'd2;
    localparam right = 2'd3;
    // Define four common rotation subsequences
    localparam A_seq = {MFDBU,MFL,MFUBD,MUR,MUR,MFDBU,MFL,MFUBD};
    localparam B_seq = {MFDBU,MFR,MFUBD,MUL,MUL,MFDBU,MFR,MFUBD};
    localparam C_seq = {MULDR,MLU,MURDL,MUR,MUR,MULDR,MLU,MURDL};
    localparam D_seq = {MULDR,MLD,MURDL,MUL,MUL,MULDR,MLD,MURDL};

    always @(*) begin
        // Determine the simpler location of the red-blue cubelet
        case(RBposition)
            {U,F,N},{F,U,N}: RBposition_simple = bot;
            {U,B,N},{B,U,N}: RBposition_simple = top;
            {U,R,N},{R,U,N}: RBposition_simple = right;
            {U,L,N},{L,U,N}: RBposition_simple = left;
            default: fail = 1;
        endcase
        // Determine the simpler location of the white-blue cubelet
        case (WBposition)
            {U,F,N},{F,U,N}: WBposition_simple = bot;
            {U,B,N},{B,U,N}: WBposition_simple = top;
            {U,R,N},{R,U,N}: WBposition_simple = right;
            {U,L,N},{L,U,N}: WBposition_simple = left;
            default: fail = 1;
        endcase

        // Based on the simplified locations of two bottom edge cubelets, determine the
        appropriate sequence of moves
        // The fail flag is set if the starting orientations of the edge cubelets are invalid
        case ({RBposition_simple,WBposition_simple})
            {bot,left}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {bot,right}: begin move_list = {B_seq}; num_moves = 6'd8; fail = 0; end
            {bot,top}: begin move_list = {A_seq}; num_moves = 6'd8; fail = 0; end
            {right,top}: begin move_list = {C_seq,A_seq}; num_moves = 6'd16; fail = 0; end
            {right,left}: begin move_list = {D_seq}; num_moves = 6'd8; fail = 0; end
            {right,bot}: begin move_list = {A_seq,D_seq}; num_moves = 6'd16; fail = 0; end
            {top,left}: begin move_list = {C_seq}; num_moves = 6'd8; fail = 0; end
            {top,right}: begin move_list = {A_seq,C_seq}; num_moves = 6'd16; fail = 0; end
            {top,bot}: begin move_list = {B_seq,C_seq}; num_moves = 6'd16; fail = 0; end
            {left,top}: begin move_list = {D_seq,A_seq}; num_moves = 6'd16; fail = 0; end
            {left,right}: begin move_list = {C_seq,B_seq}; num_moves = 6'd16; fail = 0; end
            {left,bot}: begin move_list = {A_seq,MURDL,MRD,MULDR,MUR,MUR,MURDL,MRD,MULDR};
            num_moves = 6'd16; fail = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine
// Description: Determines the rotation sequence to correct the orientation of
// all the edge cubelets on the bottom layer
//////////////////////////////////////////////////////////////////////////////////

module Bottom_Edge_Orientation(
    input [0:8] BRposition,          // position of the blue-red cubelet
    input [0:8] BWposition,          // position of the blue-white cubelet
    input [0:8] BOposition,          // position of the blue-orange cubelet
    input [0:8] BYposition,          // position of the blue-yellow cubelet
    output reg [199:0] move_list,    // rotation sequence to be performed
    output reg [5:0] num_moves,      // number of moves in the rotation sequence
    output reg fail                  // goes high if cube configuration is not valid to start
    );

    // Define the face orientations (no orientation, up, down, front, back, right, left)
    localparam N = 3'd0;
    localparam U = 3'd1;
    localparam D = 3'd2;
    localparam F = 3'd3;
    localparam B = 3'd4;
    localparam R = 3'd5;
    localparam L = 3'd6;

    // Define the 18 possible rotations
    localparam MRU = 5'd1;
    localparam MRD = 5'd2;
    localparam MFUBD = 5'd3;
    localparam MFDBU = 5'd4;
    localparam MLU = 5'd5;
    localparam MLD = 5'd6;
    localparam MFR = 5'd7;
    localparam MFL = 5'd8;
    localparam MURDL = 5'd9;
    localparam MULDR = 5'd10;
    localparam MBR = 5'd11;
    localparam MBL = 5'd12;
    localparam MUR = 5'd13;
    localparam MUL = 5'd14;
    localparam MFRBL = 5'd15;
    localparam MFLBR = 5'd16;
    localparam MDR = 5'd17;
    localparam MDL = 5'd18;

    // Define two common rotation subsequences
    localparam H_seq = {MUR,MUR,MRU,MFRBL,MBR,MBR,MFRBL,MFRBL,MFR,MUR,MUR,MFL,MFRBL,MFRBL,MBR,
    MBR,MFLBR,MRD};
    localparam G_seq = {MFL,MRD,H_seq,MRU,MFR};

    always @(*) begin
        // Based on orientations of the four bottom edge cubelets, determine the appropriate
        sequence of moves
        // The fail flag is set if the starting orientations of the edge cubelets are invalid
        case ({BRposition[0:2],BWposition[0:2],BOposition[0:2],BYposition[0:2]})
            {U,U,U,U}: begin move_list = 199'b0; num_moves = 6'd0; fail = 0; end
            {U,L,U,R}: begin move_list = {H_seq}; num_moves = 6'd18; fail = 0; end
            {F,U,B,U}: begin move_list = {MUL,H_seq,MUR}; num_moves = 6'd20; fail = 0; end
            {F,L,U,U}: begin move_list = {G_seq}; num_moves = 6'd22; fail = 0; end
            {U,L,B,U}: begin move_list = {MUL,G_seq,MUR}; num_moves = 6'd24; fail = 0; end
            {U,U,B,R}: begin move_list = {MUR,MUR,G_seq,MUR,MUR}; num_moves = 6'd26; fail = 0;
            end
            {F,U,U,R}: begin move_list = {MUR,G_seq,MUL}; num_moves = 6'd24; fail = 0; end
            {F,L,B,R}: begin move_list = {MUL,H_seq,MUR,H_seq}; num_moves = 6'd38; fail = 0; end
            default: fail = 1;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Katharine (with credit to
http://www.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html
// for the algorithm
// Description: Creates a binary coded decimal number with a tens and ones digit
//////////////////////////////////////////////////////////////////////////////////

module BCD(
    input [4:0] binary,         // binary number
    output reg [3:0] tens,      // binary coded tens digit
    output reg [3:0] ones       // binary coded ones digit
    );

    integer i;
    always @(binary) begin
        //initialize digits to zero
        tens = 4'd0;
        ones = 4'd0;

        //add 3 to columns >= 5
        for (i=4; i>=0; i=i-1) begin
            if (tens >= 5) tens = tens + 3;
            if (ones >= 5) ones = ones + 3;

            //shift left one
            tens = tens << 1;
            tens[0] = ones[3];
            ones = ones << 1;
            ones[0] = binary[i];
        end
    end
endmodule
//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// switch[7] selects between display of NTSC video and test bars
// switch[6] is used for testing the NTSC decoder
// switch[1] selects between test bar periods; these are stored to ZBT
//           during blanking periods
// switch[0] selects vertical test bars (hardwired; not stored in ZBT)
//
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09
//
// Use ramclock module to deskew clocks;  GPH
// To change display from 1024*787 to 800*600, use clock_40mhz and change
// accordingly. Verilog ntsc2zbt.v will also need changes to change resolution.
//
// Date   : 10-Nov-11

///////////////////////////////////////////////////////////////////////////////
//
// 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)
//
// 2011-Nov-10: Changed resolution to 1024 * 768.
//                   Added back ramclok to deskew RAM clock
//
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to  800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 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 zbt_6111_sample(beep, audio_reset_b,
               ac97_sdata_out, ac97_sdata_in, ac97_synch,
           ac97_bit_clock,

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

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

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

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

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

           clock_feedback_out, clock_feedback_in,

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

           rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

           mouse_clock, mouse_data, keyboard_clock, keyboard_data,

           clock_27mhz, clock1, clock2,

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

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

           switch,

           led,

           user1, user2, user3, user4,

           daughtercard,

           systemace_data, systemace_address, systemace_ce_b,
           systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
           analyzer1_data, analyzer1_clock,
           analyzer2_data, analyzer2_clock,
           analyzer3_data, analyzer3_clock,
           analyzer4_data, analyzer4_clock);

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

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

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

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

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

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

   input  clock_feedback_in;
   output clock_feedback_out;

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

   output rs232_txd, rs232_rts;
   input  rs232_rxd, rs232_cts;

   input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

   input  clock_27mhz, clock1, clock2;

   output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
   input  disp_data_in;
   output  disp_data_out;
   input  button0, button1, button2, button3, button_enter, button_right,
      button_left, button_down, button_up;
   input  [7:0] switch;
   output [7:0] led;

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

   inout [43:0] daughtercard;

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

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

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

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

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

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

   // SRAMs

/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;    // clock enable
*/

/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

/**********/

   //assign ram1_data = 36'hZ;
   // assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
    //assign ram1_clk = 1'b0;

   //These values has to be set to 0 like ram0 if ram1 is used.
//   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b0;
   assign ram1_oe_b = 1'b0;
   //assign ram1_we_b = 1'b0;
   assign ram1_bwe_b = 4'h0;

   // clock_feedback_out will be assigned by ramclock
   // assign clock_feedback_out = 1'b0;  //2011-Nov-10
   // clock_feedback_in is an input

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

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

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

   // LED Displays

//   assign disp_blank = 1'b1;
//   assign disp_clock = 1'b0;
//   assign disp_rs = 1'b0;
//   assign disp_ce_b = 1'b1;
//   assign disp_reset_b = 1'b0;
//   assign disp_data_out = 1'b0;

   // disp_data_in is an input

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

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

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

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

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

   ////////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

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

//   wire clk = clock_65mhz;  // gph 2011-Nov-10

/*   ////////////////////////////////////////////////////////////////////////////
   // Demonstration of ZBT RAM as video memory

   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 40.5MHz)
   wire clock_40mhz_unbuf,clock_40mhz;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 2
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 3
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz),.I(clock_40mhz_unbuf));

   wire clk = clock_40mhz;
*/
   wire locked;
   //assign clock_feedback_out = 0; // gph 2011-Nov-10

   ramclock rc(.ref_clock(clock_65mhz), .fpga_clock(clk),
                 .ram0_clock(ram0_clk),
                 .ram1_clock(ram1_clk),   //uncomment if ram1 is used
                 .clock_feedback_in(clock_feedback_in),
                 .clock_feedback_out(clock_feedback_out), .locked(locked));



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

   // ENTER button is user reset
   wire reset,user_reset;
   debounce db1(.reset(power_on_reset),.clock(clk),.noisy(~button_enter),.clean(user_reset));
   assign reset = user_reset | power_on_reset;

   // UP and DOWN buttons for pong paddle
   wire up,next;
   debounce db2(.reset(reset),.clock(clk),.noisy(~button_up),.clean(up));
   debounce db3(.reset(reset),.clock(clk),.noisy(~button2),.clean(next));

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

   // wire up to ZBT ram

   wire [35:0] vram_write_data;
   wire [35:0] vram0_read_data, vram1_read_data, vram_read_data, vram_read_data_render;
   wire [18:0] vram0_addr, vram1_addr, vram_addr;
   wire        vram0_we, vram1_we, vram_we, we_render;
   wire currentram;
   reg frame_hold;



   wire ram0_clk_not_used;
   zbt_6111 zbt0(clk, 1'b1, vram0_we, vram0_addr,
           vram_write_data, vram0_read_data,
           ram0_clk_not_used,   //to get good timing, don't connect ram_clk to zbt_6111
           ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

   wire ram1_clk_not_used;
   zbt_6111 zbt1(clk, 1'b1, vram1_we, vram1_addr,
           vram_write_data, vram1_read_data,
           ram1_clk_not_used,   //to get good timing, don't connect ram_clk to zbt_6111
           ram1_we_b, ram1_address, ram1_data, ram1_cen_b);

   // generate pixel value from reading ZBT memory
   wire [18:0]  vram_addr1;

   assign vram_read_data = currentram ? vram1_read_data : vram0_read_data; //read display data
   from one of the rams
   assign vram_read_data_render = currentram ? vram0_read_data : vram1_read_data;//read the
   compare depth data from the other (and write to this ram)
   wire [7:0] vr_pixel;

   vram_display vd1(reset,clk,hcount,vcount,vr_pixel,
            vram_addr1,vram_read_data);

   //render image to be displayed

   wire [4:0] rotation;
   wire [0:161] cube_config;


   serial_receive //implement the serial receive module

   serialmodule(
        .external_clk(user1[0]),
        .sync_pulse(user1[1]),
        .data(user1[2]),
        .rotation(rotation),
        .cube_config(cube_config)
   );

   wire phsync,pvsync,pblank;
   wire [10:0] x;
   wire [9:0] y;
   wire[2:0] render_state;

   //implement the 3d render module
   threeD_renderer main1(.switch(switch),.vclock(clk),.reset(reset),.frame_hold(frame_hold),
        .up(up),.next(next),.hcount(hcount),.vcount(vcount),.hsync(hsync),.vsync(vsync),.blank(
        blank),
        .phsync(phsync),.pvsync(pvsync),.pblank(pblank),.fourpixelsin(vram_read_data_render),
        .fourpixels(vram_write_data),.x(x),.y(y),.we(we_render),.currentram(currentram),.state(
        render_state),
        .cube_config(cube_config),.rotation(rotation),.complete(user1[3]));

   wire [63:0] data;
   assign data [4:0] = rotation;

   //output rotation number to hex display
   display_16hex uut (
            .reset(reset),
            .clock_27mhz(clock_27mhz),
            .data(data),
            .disp_blank(disp_blank),
            .disp_clock(disp_clock),
            .disp_rs(disp_rs),
            .disp_ce_b(disp_ce_b),
            .disp_reset_b(disp_reset_b),
            .disp_data_out(disp_data_out)
        );


   // Logic Analyzer
   // assign analyzer1_data = {render_state[2:0],currentram,we_render,vsync,y[9:0]};
   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 analyzer3_data = {12'b0,user1[3:0]};
   assign analyzer3_clock = clock_65mhz;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   // code to write pattern to ZBT memory

   wire [18:0]  vram_addr2 = (x<1024) ? {y[9:0],x[9:2]} : 19'b1;
   //vram address corresponds to pixel's screen location

   // mux selecting read/write to memory based on which write-enable is chosen
   wire      my_we = (x[1:0]==2'd3);
   wire [18:0]  write_addr = vram_addr2;

   assign   vram0_addr = currentram ? write_addr : vram_addr1;
   assign   vram1_addr = ~currentram ? write_addr : vram_addr1;

   assign   vram0_we = currentram ? we_render : 0;
   assign   vram1_we = currentram ? 0 : we_render;


   // select output pixel data
   reg [23:0]   pixel;

   parameter grey = 0;
   parameter red = 1;
   parameter orange = 2;
   parameter yellow = 3;
   parameter green = 4;
   parameter blue = 5;
   parameter white = 6;
   parameter black = 7;

   always@(*) begin
        case(vr_pixel[2:0])

                grey: pixel = 24'hAAAAAA;
                red: pixel = 24'hFF0000;
                orange: pixel = 24'hFF8000;
                yellow: pixel = 24'hFFFF00;
                green: pixel = 24'h00FF00;
                blue: pixel = 24'h0000FF;
                white: pixel = 24'hFFFFFF;
                black: pixel = 24'h000000;
                default: pixel = 24'h000000;

        endcase

   end

   reg  b,hs,vs;

   always @(posedge clk)
     begin

        b <= blank;
        hs <= hsync;
        vs <= vsync;
     end

   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clk.
   assign vga_out_red = pixel[23:16];
   assign vga_out_green = pixel[15:8];
   assign vga_out_blue = pixel[7:0];
   assign vga_out_sync_b = 1'b1;    // not used
   assign vga_out_pixel_clock = ~clk;
   assign vga_out_blank_b = ~b;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   // debugging
   assign led = ~{vram0_addr[18:13],reset,switch[0]};

endmodule

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

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

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

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

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

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

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

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

////////////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.
//
// Bug due to memory management will be fixed. The bug happens because
// memory is called based on current hcount & vcount, which will actually
// shows up 2 cycle in the future. Not to mention that these incoming data
// are latched for 2 cycles before they are used. Also remember that the
// ntsc2zbt's addressing protocol has been fixed.

// The original bug:
// -. At (hcount, vcount) = (100, 201) data at memory address(0,100,49)
//    arrives at vram_read_data, latch it to vr_data_latched.
// -. At (hcount, vcount) = (100, 203) data at memory address(0,100,49)
//    is latched to last_vr_data to be used for display.
// -. Remember that memory address(0,100,49) contains camera data
//    pixel(100,192) - pixel(100,195).
// -. At (hcount, vcount) = (100, 204) camera pixel data(100,192) is shown.
// -. At (hcount, vcount) = (100, 205) camera pixel data(100,193) is shown.
// -. At (hcount, vcount) = (100, 206) camera pixel data(100,194) is shown.
// -. At (hcount, vcount) = (100, 207) camera pixel data(100,195) is shown.
//
// Unfortunately this means that at (hcount == 0) to (hcount == 11) data from
// the right side of the camera is shown instead (including possible sync signals).

// To fix this, two corrections has been made:
// -. Fix addressing protocol in ntsc_to_zbt module.
// -. Forecast hcount & vcount 8 clock cycles ahead and use that
//    instead to call data from ZBT.


module vram_display(reset,clk,hcount,vcount,vr_pixel,
            vram_addr,vram_read_data);

   input reset, clk;
   input [10:0] hcount;
   input [9:0]  vcount;
   output [7:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;
   //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
   wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
   wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) : vcount;

//   wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount);
//   wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount) : vcount;
//
   wire [18:0]   vram_addr = {1'b0, vcount_f, hcount_f[9:2]};

   wire [1:0]    hc4 = hcount[1:0];
   reg [7:0]     vr_pixel;
   reg [35:0]    vr_data_latched;
   reg [35:0]    last_vr_data;

   always @(posedge clk)
     last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;

   always @(posedge clk)
     vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;

   always @(*)      // each 36-bit word from RAM is decoded to 4 bytes
     case (hc4)
       2'd3: vr_pixel = last_vr_data[7:0];
       2'd2: vr_pixel = last_vr_data[7+8:0+8];
       2'd1: vr_pixel = last_vr_data[7+16:0+16];
       2'd0: vr_pixel = last_vr_data[7+24:0+24];
     endcase

endmodule // vram_display

///////////////////////////////////////////////////////////////////////////////
// parameterized delay line

module delayN(clk,in,out);
   input clk;
   input in;
   output out;

   parameter NDELAY = 3;

   reg [NDELAY-1:0] shiftreg;
   wire         out = shiftreg[NDELAY-1];

   always @(posedge clk)
     shiftreg <= {shiftreg[NDELAY-2:0],in};

endmodule // delayN

//////////////////////////////////////////////////////////////////////////////
// ramclock module

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

```
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
///////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
            clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                 // Reference clock input
   output fpga_clock;               // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
   input  clock_feedback_in;        // Output to feedback trace
   output clock_feedback_out;       // Input from feedback trace
   output locked;                   // Indicates that clock outputs are stable

   wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

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

   //To force ISE to compile the ramclock, this line has to be removed.
   //IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

    assign ref_clk = ref_clock;

   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

   DCM int_dcm (.CLKFB(fpga_clock),
        .CLKIN(ref_clk),
        .RST(dcm_reset),
        .CLK0(fpga_clk),
        .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

   BUFG ext_buf (.O(ram_clock), .I(ram_clk));

   IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

   DCM ext_dcm (.CLKFB(fb_clk),
            .CLKIN(ref_clk),
            .RST(dcm_reset),
            .CLK0(ram_clk),
            .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

   SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
             .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   // synthesis attribute init of dcm_rst_sr is "000F";


   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
              .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
              .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
              .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;

endmodule
```

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

/////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
          ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;                    // system clock
   input cen;                    // clock enable for gating ZBT cycles
   input we;                     // write enable (active HIGH)
   input [18:0] addr;            // memory address
   input [35:0] write_data;      // data to write
   output [35:0] read_data;      // data read from memory
   output    ram_clk;            // physical line to ram clock
   output    ram_we_b;           // physical line to ram we_b
   output [18:0] ram_address;    // physical line to ram address
   inout [35:0]  ram_data;       // physical line to ram data
   output    ram_cen_b;          // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
   wire      ram_cen_b = ~cen;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0]   we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;

   // create two-stage pipeline for write data

   reg [35:0]  write_data_old1;
   reg [35:0]  write_data_old2;
   always @(posedge clk)
     if (cen)
       {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

   // wire to ZBT RAM signals

   assign      ram_we_b = ~we;
   assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                                // set to zero as place holder

//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                                   // times if its clk edges equal FPGA's
                                   // so we clock it on the falling edges
                                   // and thus let data stabilize longer
   assign      ram_address = addr;

   assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
   assign      read_data = ram_data;

endmodule // zbt_6111
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Jack
//////////////////////////////////////////////////////////////////////////////////

module threeD_renderer (
   input [7:0] switch,          //for debugging
   input vclock,                // 65MHz clock
   input reset,                 // 1 to initialize module
   input frame_hold,            // not used
   input up,                    //not used
   input next,                  // high to start next rotation
   input [10:0] hcount,         // horizontal index of current pixel (0..1023)
   input [9:0] vcount,          // vertical index of current pixel (0..767)
   input hsync,                 // XVGA horizontal sync signal (active low)
   input vsync,                 // XVGA vertical sync signal (active low)
   input blank,                 // XVGA blanking (1 means output black pixel)
   output phsync,               // horizontal sync
   output pvsync,               // vertical sync
   output pblank,               // blanking
   input [31:0] fourpixelsin,
   output [31:0] fourpixels,    //four byte (1 per pixel) to be sent to ZBT RAM
   output [10:0] x,             //horizontal pixel co-ord
   output [9:0] y,              //vertical pixel co-ordinates
   output reg we,               //ZBT write enable
   output reg currentram,       //which RAM to be written to
   output reg [2:0] state,      //for debugging
   input [0:161] cube_config,   //cube configuration
   input [4:0] rotation,        //rotation type
   output complete              //tell solve cube block that animation is done
   );


    parameter facewidth = 100; //puck parameters
    parameter faceheight = 100;
    parameter xcubecenter = (1024/2);
    parameter ycubecenter = (768/2);

    parameter grey = 0;
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    parameter idle = 0;
    parameter clearing = 1;
    parameter readingtop = 2;
    parameter writingtop = 3;
    parameter readingleft = 4;
    parameter writingleft = 5;
    parameter readingright = 6;
    parameter writingright = 7;

    parameter cubelet_width = 90;
    parameter border_thickness = 5;

    parameter xpos1 = 0;
    parameter xpos2 = cubelet_width + (2*border_thickness);
    parameter xpos3 = 2*(cubelet_width + (2*border_thickness));

    parameter ypos1 = 0;
    parameter ypos2 = cubelet_width + (2*border_thickness);
    parameter ypos3 = 2*(cubelet_width + (2*border_thickness));

    parameter face_pipeline_delay = 0;


    //some of these are no longer used
    wire[2:0] colourout,colourouttop,colouroutleft,colouroutright;
    wire[2:0] colouroutt1,colouroutt2,colouroutt3,colouroutt4,colouroutt5,colouroutt6,
    colouroutt7,colouroutt8,colouroutt9;
    wire[2:0] colouroutl1,colouroutl2,colouroutl3,colouroutl4,colouroutl5,colouroutl6,
    colouroutl7,colouroutl8,colouroutl9;
    wire[2:0] colouroutr1,colouroutr2,colouroutr3,colouroutr4,colouroutr5,colouroutr6,
    colouroutr7,colouroutr8,colouroutr9;
    reg[10:0] xcount;
    reg [9:0] ycount,zcount;
    reg clear_bit;
    reg [1:0] clk_delay;
    reg [10:0] clear_addrx;
    reg [9:0] clear_addry;
    reg [31:0] fourpixels_nc;

    wire [11:0] xprime_unsigned_with_signed_bit;
    wire [10:0] xprime_unsigned;
    reg [10:0] xprime, xfactor1, xfactor2;
    wire [10:0] xprimet_signed, xprimetop, xprimeleft, xprimeright;
    wire [10:0] xprimet1, xprimet2, xprimet3, xprimet4, xprimet5, xprimet6, xprimet7, xprimet8,
    xprimet9;
    wire [10:0] xprimel1, xprimel2, xprimel3, xprimel4, xprimel5, xprimel6, xprimel7, xprimel8,
    xprimel9;
    wire [10:0] xprimer1, xprimer2, xprimer3, xprimer4, xprimer5, xprimer6, xprimer7, xprimer8,
    xprimer9;

    wire [11:0] yprime_unsigned_with_signed_bit;
    wire [9:0] yprime_unsigned;
    reg[9:0] yprime, yfactor1, yfactor2;
    wire [9:0] yprimet_signed, yprimetop, yprimeleft, yprimeright;
    wire [9:0] yprimet1, yprimet2, yprimet3, yprimet4, yprimet5, yprimet6, yprimet7, yprimet8,
    yprimet9;
    wire [9:0] yprimel1, yprimel2, yprimel3, yprimel4, yprimel5, yprimel6, yprimel7, yprimel8,
    yprimel9;
    wire [9:0] yprimer1, yprimer2, yprimer3, yprimer4, yprimer5, yprimer6, yprimer7, yprimer8,
    yp.rimer9;

    wire [11:0] zprime_unsigned_with_signed_bit;
    wire [9:0] zprime;
    reg[9:0] /*zprime,*/ zfactor1, zfactor2;
    wire [9:0] zprimet_signed, zprimetop, zprimeleft, zprimeright;
    wire [9:0] zprimet1, zprimet2, zprimet3, zprimet4, zprimet5, zprimet6, zprimet7, zprimet8,
    zprimet9;
    wire [9:0] zprimel1, zprimel2, zprimel3, zprimel4, zprimel5, zprimel6, zprimel7, zprimel8,
    zprimel9;
    wire [9:0] zprimer1, zprimer2, zprimer3, zprimer4, zprimer5, zprimer6, zprimer7, zprimer8,
    zprimer9;


    reg[4:0] zrelative;

    wire cbtop, cbt1, cbt2, cbt3, cbt4, cbt5, cbt6, cbt7, cbt8, cbt9;
    wire cbleft, cbl1, cbl2, cbl3, cbl4, cbl5, cbl6, cbl7, cbl8, cbl9;
    wire cbright, cbr1, cbr2, cbr3, cbr4, cbr5, cbr6, cbr7, cbr8, cbr9;

    reg [2:0] colourout_pipeline0, colourout_pipeline1;

    wire[1:0] rott1,rott2,rott3,rott4,rott5,rott6,rott7,rott8,rott9;
    wire[1:0] rotl1,rotl2,rotl3,rotl4,rotl5,rotl6,rotl7,rotl8,rotl9;
    wire[1:0] rotr1,rotr2,rotr3,rotr4,rotr5,rotr6,rotr7,rotr8,rotr9;
    wire signed[17:0] cosfactor, sinfactor;


    wire [0:161] displayed_cube_config;

   //implement rotator module
    rotator main_rotator(
     .vclock(vclock),
     .start(next),
     .rotation(rotation),
     .rott1(rott1),.rott2(rott2),.rott3(rott3),.rott4(rott4),.rott5(rott5),.rott6(rott6),.rott7(
     rott7),.rott8(rott8),.rott9(rott9),
     .rotl1(rotl1),.rotl2(rotl2),.rotl3(rotl3),.rotl4(rotl4),.rotl5(rotl5),.rotl6(rotl6),.rotl7(
     rotl7),.rotl8(rotl8),.rotl9(rotl9),
     .rotr1(rotr1),.rotr2(rotr2),.rotr3(rotr3),.rotr4(rotr4),.rotr5(rotr5),.rotr6(rotr6),.rotr7(
     rotr7),.rotr8(rotr8),.rotr9(rotr9),
     .cosfactor(cosfactor), .sinfactor(sinfactor),
     .complete(complete),
     .cube_config(cube_config),
     .displayed_cube_config(displayed_cube_config)
    );


    //implement the 9 top faces
    top_face facet1(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos1),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[24:26]),.colourout(colouroutt1),.xprime(
    xprimet1),.yprime (yprimet1),.zprime (zprimet1),.control_bit(cbt1),.rotation(rott1),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    top_face facet2(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos1),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[21:23]),.colourout(colouroutt2),.xprime(
    xprimet2),.yprime (yprimet2),.zprime (zprimet2),.control_bit(cbt2),.rotation(rott2),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    top_face facet3(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos1),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[18:20]),.colourout(colouroutt3),.xprime(
    xprimet3),.yprime (yprimet3),.zprime (zprimet3),.control_bit(cbt3),.rotation(rott3),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    top_face facet4(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos2),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[15:17]),.colourout(colouroutt4),.xprime(
    xprimet4),.yprime (yprimet4),.zprime (zprimet4),.control_bit(cbt4),.rotation(rott4),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    top_face facet5(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos2),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[12:14]),.colourout(colouroutt5),.xprime(
    xprimet5),.yprime (yprimet5),.zprime (zprimet5),.control_bit(cbt5),.rotation(rott5),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    top_face facet6(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos2),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[9:11]),.colourout(colouroutt6),.xprime(xprimet6
    ),.yprime (yprimet6),.zprime (zprimet6),.control_bit(cbt6),.rotation(rott6),.cosfactor(
    cosfactor),.sinfactor(sinfactor));
    top_face facet7(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos3),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[6:8]),.colourout(colouroutt7),.xprime(xprimet7
    ),.yprime (yprimet7),.zprime (zprimet7),.control_bit(cbt7),.rotation(rott7),.cosfactor(
    cosfactor),.sinfactor(sinfactor));
    top_face facet8(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos3),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[3:5]),.colourout(colouroutt8),.xprime(xprimet8
    ),.yprime (yprimet8),.zprime (zprimet8),.control_bit(cbt8),.rotation(rott8),.cosfactor(
    cosfactor),.sinfactor(sinfactor));
    top_face facet9(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos3),.vcount(ycount),.z(0),.
    dcount(zcount),.colour(displayed_cube_config[0:2]),.colourout(colouroutt9),.xprime(xprimet9
    ),.yprime (yprimet9),.zprime (zprimet9),.control_bit(cbt9),.rotation(rott9),.cosfactor(
    cosfactor),.sinfactor(sinfactor));

   //choose which face module to actually send info from based on the control bit of each
    assign colourouttop = cbt1 ? colouroutt1 : cbt2 ? colouroutt2 :cbt3 ? colouroutt3 : cbt4 ?
    colouroutt4 : cbt5 ? colouroutt5 :cbt6 ? colouroutt6 : cbt7 ? colouroutt7 : cbt8 ?
    colouroutt8 :cbt9 ? colouroutt9 :3'd7;

   //choose which coordinates to use based on the control bit
    assign xprimetop = cbt1 ? xprimet1 : cbt2 ? xprimet2 : cbt3 ? xprimet3 : cbt4 ? xprimet4 :
    cbt5 ? xprimet5 : cbt6 ? xprimet6 : cbt7 ? xprimet7 : cbt8 ? xprimet8 : cbt9 ? xprimet9 :
    1024;
    assign yprimetop = cbt1 ? yprimet1 : cbt2 ? yprimet2 : cbt3 ? yprimet3 : cbt4 ? yprimet4 :
    cbt5 ? yprimet5 : cbt6 ? yprimet6 : cbt7 ? yprimet7 : cbt8 ? yprimet8 : cbt9 ? yprimet9 :
    768;
    assign zprimetop = cbt1 ? zprimet1 : cbt2 ? zprimet2 : cbt3 ? zprimet3 : cbt4 ? zprimet4 :
    cbt5 ? zprimet5 : cbt6 ? zprimet6 : cbt7 ? zprimet7 : cbt8 ? zprimet8 : cbt9 ? zprimet9 :
    768;

    assign cbtop = cbt1 || cbt2 || cbt3 || cbt4 || cbt5 || cbt6 || cbt7 || cbt8 || cbt9;

    //implement the 9 left faces
    left_face facel1(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos1),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[60:62]),.colourout(colouroutl1),.xprime(
    xprimel1),.yprime (yprimel1),.zprime (zprimel1),.control_bit(cbl1),.rotation(rotl1),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel2(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos1),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[57:59]),.colourout(colouroutl2),.xprime(
    xprimel2),.yprime (yprimel2),.zprime (zprimel2),.control_bit(cbl2),.rotation(rotl2),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel3(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos1),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[54:56]),.colourout(colouroutl3),.xprime(
    xprimel3),.yprime (yprimel3),.zprime (zprimel3),.control_bit(cbl3),.rotation(rotl3),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel4(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos2),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[69:71]),.colourout(colouroutl4),.xprime(
    xprimel4),.yprime (yprimel4),.zprime (zprimel4),.control_bit(cbl4),.rotation(rotl4),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel5(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos2),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[66:68]),.colourout(colouroutl5),.xprime(
    xprimel5),.yprime (yprimel5),.zprime (zprimel5),.control_bit(cbl5),.rotation(rotl5),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel6(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos2),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[63:65]),.colourout(colouroutl6),.xprime(
    xprimel6),.yprime (yprimel6),.zprime (zprimel6),.control_bit(cbl6),.rotation(rotl6),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel7(.vclock(vclock),.x(xpos1),.hcount(xcount),.y(ypos3),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[78:80]),.colourout(colouroutl7),.xprime(
    xprimel7),.yprime (yprimel7),.zprime (zprimel7),.control_bit(cbl7),.rotation(rotl7),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel8(.vclock(vclock),.x(xpos2),.hcount(xcount),.y(ypos3),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[75:77]),.colourout(colouroutl8),.xprime(
    xprimel8),.yprime (yprimel8),.zprime (zprimel8),.control_bit(cbl8),.rotation(rotl8),.
    cosfactor(cosfactor),.sinfactor(sinfactor));
    left_face facel9(.vclock(vclock),.x(xpos3),.hcount(xcount),.y(ypos3),.vcount(zcount),.z(0),.
    dcount(ycount),.colour(displayed_cube_config[72:74]),.colourout(colouroutl9),.xprime(
    xprimel9),.yprime (yprimel9),.zprime (zprimel9),.control_bit(cbl9),.rotation(rotl9),.
    cosfactor(cosfactor),.sinfactor(sinfactor));

    //choose which face module to actually send info from based on the control bit of each
    assign colouroutleft = cbl1 ? colouroutl1 : cbl2 ? colouroutl2 :cbl3 ? colouroutl3 : cbl4 ?
    colouroutl4 : cbl5 ? colouroutl5 :cbl6 ? colouroutl6 : cbl7 ? colouroutl7 : cbl8 ?
    colouroutl8 :cbl9 ? colouroutl9 :3'd7;

    //choose which coordinates to use based on the control bit
    assign xprimeleft = cbl1 ? xprimel1 : cbl2 ? xprimel2 : cbl3 ? xprimel3 : cbl4 ? xprimel4 :
    cbl5 ? xprimel5 : cbl6 ? xprimel6 : cbl7 ? xprimel7 : cbl8 ? xprimel8 : cbl9 ? xprimel9 :
    1024;
    assign yprimeleft = cbl1 ? yprimel1 : cbl2 ? yprimel2 : cbl3 ? yprimel3 : cbl4 ? yprimel4 :
    cbl5 ? yprimel5 : cbl6 ? yprimel6 : cbl7 ? yprimel7 : cbl8 ? yprimel8 : cbl9 ? yprimel9 :
    768;
    assign zprimeleft = cbl1 ? zprimel1 : cbl2 ? zprimel2 : cbl3 ? zprimel3 : cbl4 ? zprimel4 :
    cbl5 ? zprimel5 : cbl6 ? zprimel6 : cbl7 ? zprimel7 : cbl8 ? zprimel8 : cbl9 ? zprimel9 :
    768;

    assign cbleft = cbl1 || cbl2 || cbl3 || cbl4 || cbl5 || cbl6 || cbl7 || cbl8 || cbl9;
```

```
//implement the 9 right faces
right_face facer1(.vclock(vclock),.x(xpos1),.hcount(ycount),.y(ypos1),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[81:83]),.colourout(colouroutr1),.xprime(
xprimer1),.yprime (yprimer1),.zprime (zprimer1),.control_bit(cbr1),.rotation(rotr1),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer2(.vclock(vclock),.x(xpos2),.hcount(ycount),.y(ypos1),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[84:86]),.colourout(colouroutr2),.xprime(
xprimer2),.yprime (yprimer2),.zprime (zprimer2),.control_bit(cbr2),.rotation(rotr2),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer3(.vclock(vclock),.x(xpos3),.hcount(ycount),.y(ypos1),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[87:89]),.colourout(colouroutr3),.xprime(
xprimer3),.yprime (yprimer3),.zprime (zprimer3),.control_bit(cbr3),.rotation(rotr3),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer4(.vclock(vclock),.x(xpos1),.hcount(ycount),.y(ypos2),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[90:92]),.colourout(colouroutr4),.xprime(
xprimer4),.yprime (yprimer4),.zprime (zprimer4),.control_bit(cbr4),.rotation(rotr4),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer5(.vclock(vclock),.x(xpos2),.hcount(ycount),.y(ypos2),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[93:95]),.colourout(colouroutr5),.xprime(
xprimer5),.yprime (yprimer5),.zprime (zprimer5),.control_bit(cbr5),.rotation(rotr5),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer6(.vclock(vclock),.x(xpos3),.hcount(ycount),.y(ypos2),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[96:98]),.colourout(colouroutr6),.xprime(
xprimer6),.yprime (yprimer6),.zprime (zprimer6),.control_bit(cbr6),.rotation(rotr6),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer7(.vclock(vclock),.x(xpos1),.hcount(ycount),.y(ypos3),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[99:101]),.colourout(colouroutr7),.xprime(
xprimer7),.yprime (yprimer7),.zprime (zprimer7),.control_bit(cbr7),.rotation(rotr7),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer8(.vclock(vclock),.x(xpos2),.hcount(ycount),.y(ypos3),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[102:104]),.colourout(colouroutr8),.xprime(
xprimer8),.yprime (yprimer8),.zprime (zprimer8),.control_bit(cbr8),.rotation(rotr8),.
cosfactor(cosfactor),.sinfactor(sinfactor));
right_face facer9(.vclock(vclock),.x(xpos3),.hcount(ycount),.y(ypos3),.vcount(zcount),.z(0
),.dcount(xcount),.colour(displayed_cube_config[105:107]),.colourout(colouroutr9),.xprime(
xprimer9),.yprime (yprimer9),.zprime (zprimer9),.control_bit(cbr9),.rotation(rotr9),.
cosfactor(cosfactor),.sinfactor(sinfactor));

//choose which face module to actually send info from based on the control bit of each
assign colouroutright = cbr1 ? colouroutr1 : cbr2 ? colouroutr2 :cbr3 ? colouroutr3 : cbr4 ?
 colouroutr4 : cbr5 ? colouroutr5 :cbr6 ? colouroutr6 : cbr7 ? colouroutr7 : cbr8 ?
colouroutr8 :cbr9 ? colouroutr9 :3'd7;

//choose which coordinates to use based on the control bit
assign xprimeright = cbr1 ? xprimer1 : cbr2 ? xprimer2 : cbr3 ? xprimer3 : cbr4 ? xprimer4 :
 cbr5 ? xprimer5 : cbr6 ? xprimer6 : cbr7 ? xprimer7 : cbr8 ? xprimer8 : cbr9 ? xprimer9 :
1024;
assign yprimeright = cbr1 ? yprimer1 : cbr2 ? yprimer2 : cbr3 ? yprimer3 : cbr4 ? yprimer4 :
 cbr5 ? yprimer5 : cbr6 ? yprimer6 : cbr7 ? yprimer7 : cbr8 ? yprimer8 : cbr9 ? yprimer9 :
768;
assign zprimeright = cbr1 ? zprimer1 : cbr2 ? zprimer2 : cbr3 ? zprimer3 : cbr4 ? zprimer4 :
 cbr5 ? zprimer5 : cbr6 ? zprimer6 : cbr7 ? zprimer7 : cbr8 ? zprimer8 : cbr9 ? zprimer9 :
768;

assign cbright = cbr1 || cbr2 || cbr3 || cbr4 || cbr5 || cbr6 || cbr7 || cbr8 || cbr9;

//choose which face module to actually send info from based on the control bit of each
assign colourout = (cbtop) ? colourouttop : (cbleft) ? colouroutleft :(cbright) ?
colouroutright : 3'd7;

//choose which coordinates to use based on the control bit
assign xprime_unsigned = cbtop ? xprimetop : cbleft ? xprimeleft : cbright ? xprimeright :
1024;
assign yprime_unsigned = cbtop ? yprimetop : cbleft ? yprimeleft : cbright ? yprimeright :
768;
assign zprime = cbtop ? zprimetop : cbleft ? zprimeleft : cbright ? zprimeright : 768;

// if not clearing the display set the x an y coords to those calculated by the face module
assign x = clear_bit? clear_addrx : xprime_unsigned;
assign y = clear_bit? clear_addry : yprime_unsigned;


assign fourpixels = clear_bit ? 32'hFFFF_FFFF : fourpixels_nc; //set the 4 bytes sent to
the ZBT to black if clearing, otherwise set them to the calculated values

always @(posedge vclock) begin
    case(state)
        idle:   begin //wait to start determining new frame
            if(vsync) begin //(vsync) begin
                    xcount <= 0;
                    ycount <= 0;
                    zcount <= 0;
                    clear_addrx <= 0;
                    clear_addry <= 0;
                    clear_bit <= 1;
                    state <= clearing;
                    we<=1;
                end
            end

        clearing:begin // before determining and drawing new frame to ZBT clear out old ZBT
        data
                        if(clear_addrx >= 1024) begin
                            clear_addrx <= 0;
                            if(clear_addry >= 767)begin
                                clear_addry <= 0;
                                we <= 0;
                                state <= readingtop;
                                clear_bit <= 0;
                                clk_delay <= 0;
                            end
                            else clear_addry <= clear_addry + 1;
                        end
                        else clear_addrx <= clear_addrx + 4;
                    end

        readingtop: begin //delay so ZBT has time be read and written to

                        case (clk_delay)

                        0: begin
                                we<=0;
                                clk_delay <= 1;
                            end
                        1: begin
                                we<=0;
                                clk_delay <= 2;
                            end
                        2: begin
                                we<=1;
                                clk_delay <= 3;
                            end
                        3: begin
                                //we <= 0;
                                if(zprime[8:4] <= zrelative) we <=1;//write only if pixel
                                would be visible
                                else we <= 0;
                                clk_delay <= 0;
                                state <= writingtop;
                            end
                        default: clk_delay <= 0;
                        endcase


                    end


        writingtop: begin // determine top face pixels
                        if(xcount == 300) begin
                            xcount <= 0;
                            if(ycount == 300) begin
                                ycount <= 0;
                                state<= readingleft;

                            end
                            else ycount <= ycount +1;
                        end
                        else begin
                            xcount <= xcount+1;
                            zcount<= 0;
                            state<=readingtop;
                            clk_delay <= 0;
                            we <= 0;
                        end

                    end
```

```
            readingleft: begin //delay so ZBT has time be read and written to

                    case (clk_delay)

                    0: begin
                            we<=0;
                            clk_delay <= 1;
                        end
                    1: begin
                            we<=0;
                            clk_delay <= 2;
                        end
                    2: begin
                            we<=1;
                            clk_delay <= 3;
                        end
                    3: begin
                            //we <= 0;
                            if(zprime[8:4] <= zrelative) we <=1;//write only if pixel
                            would be visible
                            else we <= 0;
                            clk_delay <= 0;
                            state <= writingleft;
                        end
                    default: clk_delay <= 0;
                    endcase


                end


            writingleft: begin //determine left face pixels
                    if(xcount == 300) begin
                        xcount <= 0;
                        if(zcount == 300) begin
                            zcount <= 0;
                            state<= readingright;

                        end
                        else zcount <= zcount +1;
                    end
                    else begin
                        xcount <= xcount+1;
                        ycount<= 0;
                        state<=readingleft;
                        clk_delay <= 0;
                        we <= 0;
                    end

                end

            readingright: begin //delay so ZBT has time be read and written to
```

```
                    case (clk_delay)

                    0: begin
                            we<=0;
                            clk_delay <= 1;
                        end
                    1: begin
                            we<=0;
                            clk_delay <= 2;
                        end
                    2: begin
                            //we<=1;
                            if(zprime[8:4] <= zrelative) we <=1;//write only if pixel
                            would be visible
                            else we <= 0;
                            clk_delay <= 3;
                        end
                    3: begin
                            we <= 0;
                            clk_delay <= 0;
                            state <= writingright;
                        end
                    default: clk_delay <= 0;
                    endcase




                end


            writingright: begin // determine right face pixels
                    if(ycount == 300) begin
                        ycount <= 0;
                        if(zcount == 300) begin
                            zcount <= 0;
                            state<= idle;
                            currentram<=~currentram; //switch RAM when full frame has
                            been rendered
                        end
                        else zcount <= zcount +1;
                    end
                    else begin
                        ycount <= ycount+1;
                        xcount<= 0;
                        state<=readingright;
                        clk_delay <= 0;
                        we <= 0;
                    end

                end


            default: state <= idle;
```

```
        endcase

    end

    wire [1:0]   hc4 = x[1:0];

    always @(*) begin   // 4 bytes encoded to one word

        // store a combo of depth and colour in each byte
        case (hc4)
        2'd3:       begin
                        fourpixels_nc[7:0] = {zprime[8:4],colourout};
                        fourpixels_nc[7+8:0+8] = fourpixelsin[7+8:0+8];
                        fourpixels_nc[7+16:0+16] = fourpixelsin[7+16:0+16];
                        fourpixels_nc[7+24:0+24] = fourpixelsin[7+24:0+24];
                        zrelative = fourpixelsin[7:3];
                    end

        2'd2:       begin
                        fourpixels_nc[7:0] = fourpixels[7:0];
                        fourpixels_nc[7+8:0+8] = {zprime[8:4],colourout};
                        fourpixels_nc[7+16:0+16] = fourpixelsin[7+16:0+16];
                        fourpixels_nc[7+24:0+24] = fourpixelsin[7+24:0+24];
                        zrelative = fourpixelsin[7+8:3+8];
                    end

        2'd1:       begin
                        fourpixels_nc[7:0] = fourpixels[7:0];
                        fourpixels_nc[7+8:0+8] = fourpixelsin[7+8:0+8];
                        fourpixels_nc[7+16:0+16] = {zprime[8:4],colourout};
                        fourpixels_nc[7+24:0+24] = fourpixelsin[7+24:0+24];
                        zrelative = fourpixelsin[7+16:3+16];
                    end

        2'd0:       begin
                        fourpixels_nc[7:0] = fourpixels[7:0];
                        fourpixels_nc[7+8:0+8] = fourpixelsin[7+8:0+8];
                        fourpixels_nc[7+16:0+16] = fourpixelsin[7+16:0+16];
                        fourpixels_nc[7+24:0+24] = {zprime[8:4],colourout};
                        zrelative = fourpixelsin[7+24:3+24];
                    end
     endcase

    end

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Jack
//////////////////////////////////////////////////////////////////////////////////

module top_face
   #(parameter WIDTH = 90,                         // default width: 90 pixels
               border_thickness = 5)               // default border thickness: 100 pixels
   (input wire vclock,
     input wire [10:0]    x,hcount,     //x = top left hand x co-ordinate, hcount = current
     horizontal pixel being calculated
     input wire [9:0]     y,vcount,     //y = top left hand y co-ordinate, vcount = current
     vertical pixel being calculated
     input wire [9:0]     z,dcount,     //z = top left hand z co-ordinate, dcount = current
     depth pixel being calculated
     input wire [2:0]     colour,       //colour of face
     output reg [2:0]     colourout,    //current pixel colour
     output reg [10:0]    xprime,       //mapped x co-ordinate
     output reg [9:0]     yprime,       //mapped y co-ordinate
     output reg [9:0]     zprime,       //mapped z co-ordinate
     output reg           control_bit,  //bit to determine if this face is the one currently
     being rendered
     input wire [1:0]     rotation,     //rotation to be performed
     input wire signed [17:0] cosfactor, sinfactor); //sin and cos of angle by which face is
     rotated

    parameter grey = 0;
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    parameter xcubecenter = (1024/2);
    parameter ycubecenter = (768/2);


    reg signed [11:0] xmatrixA, xmatrixB, xmatrixC, xmatrixD, xmatrixE, xprimeAsigned;


    reg signed [10:0] ymatrixA, ymatrixB, ymatrixC, ymatrixD, ymatrixE, yprimeAsigned;


    reg signed [10:0] zmatrixA, zmatrixB, zmatrixC, zmatrixD, zmatrixE, zmatrixF, zprimeAsigned;

    reg [2:0] colouroutp1, colouroutp2, colouroutp3, colouroutp4, colouroutp5;
    reg control_bitp1, control_bitp2, control_bitp3, control_bitp4, control_bitp5;


    always @ * begin //basically just the blob module
```

```
        if (((hcount >= (x + border_thickness)) && (hcount < (x+WIDTH+border_thickness))) && ((
        vcount >= (y + border_thickness)) && (vcount < (y+WIDTH + border_thickness))) && (dcount
         == z)) colouroutp1 = colour;
        else colouroutp1 = black;

        if (((hcount >= (x+1)) && (hcount < (x+WIDTH))) && ((vcount >= (y+1)) && (vcount < (y+
        WIDTH))) && (dcount == z)) control_bitp1 = 1;
        else control_bitp1 = 0;

    end

    always @(posedge vclock) begin

        case(rotation)

        1: begin // rotation about z axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //xcosA
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //xsinA
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17); //ycosA
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17); //ysinA
            zmatrixA <= $unsigned(dcount); //zcosA (=1)
            zmatrixB <= 0; //zsinA(=0)
            colouroutp2 <= colouroutp1; //pipeline
            control_bitp2 <= control_bitp1; //pipeline

            xmatrixC <= xmatrixA - ymatrixB + 150; // xcosA - ysinA
            ymatrixC <= xmatrixB + ymatrixA + 150; // xsinA + ycosA
            zmatrixC <= zmatrixA; // pipeline
            colouroutp3 <= colouroutp2; //pipeline
            control_bitp3 <= control_bitp2; //pipeline

            end

        2:  begin //rotation about y axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //xcosB
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //xsinB
            ymatrixA <= $unsigned(vcount); //ycosB
            ymatrixB <= 0; //ysinB
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17); //zcosB
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17); //zsinB
            colouroutp2 <= colouroutp1; //pipeline
            control_bitp2 <= control_bitp1; //pipeline

            xmatrixC <= xmatrixA + zmatrixB + 150; //xcosB + zsinB
            ymatrixC <= ymatrixA; //pipeline
            zmatrixC <= zmatrixA - xmatrixB +150; // -xinsB + zcosB
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end
        3: begin // rotation about x axis

            xmatrixA <= $unsigned(hcount);//becoming tedious to comment... similar to above...
            xmatrixB <= 0;
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17);
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17);
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17);
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17);
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA - zmatrixB + 150;
            zmatrixC <= ymatrixB + zmatrixA + 150;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        0: begin //no rotation


           //two pipeline stages
            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0;
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        default: begin //no rotation

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0;
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        endcase

       //map to 2D screen coords
        xmatrixD <= (xmatrixC*113512)>>>17; //rotated x-coord * cos30
        ymatrixD <= (ymatrixC*113512)>>>17; //rotated y-coord * cos30
        xmatrixE <= (xmatrixC)>>>1; //rotated x-coord * sin30
        ymatrixE <= (ymatrixC)>>>1; //rotated y-coord * sin30
        zmatrixD <= (xmatrixC*92681)>>>17; //rotated x-coord * cos45
        zmatrixE <= (ymatrixC*92681)>>>17; //rotated y-coord * sin54
        zmatrixF <= zmatrixC; // pipeline
        colouroutp4 <= colouroutp3; // pipeline
        control_bitp4 <= control_bitp3; // pipeline

        xprimeAsigned <= xcubecenter + ymatrixD - xmatrixD; //offset + ycos30 - xcos30
        yprimeAsigned <= zmatrixF + ycubecenter - xmatrixE - ymatrixE; //offset - xsin30 -
        ysin30 + z
        zprimeAsigned <= zmatrixE + zmatrixD + 86; // xcos45 + ysin45 + offset
        colouroutp5 <= colouroutp4;
        control_bitp5 <= control_bitp4;

        xprime<= xprimeAsigned[10:0];
        yprime<= yprimeAsigned[10:0];
        zprime<= zprimeAsigned[10:0];
        colourout <= colouroutp5;
        control_bit <= control_bitp5;




    end
endmodule

//almost identical to top_face => same comments!
module left_face
   #(parameter WIDTH = 90,            // default width: 90 pixels
               border_thickness = 5)
   (input wire vclock,
     input wire [10:0] x,hcount,
    input wire[9:0] y,vcount,
     input wire[9:0] z,dcount,
    input wire [2:0] colour,
     output reg [2:0] colourout,
     output reg [10:0] xprime,
     output reg [9:0] yprime,
     output reg [9:0] zprime,
     output reg control_bit,
     input wire [1:0] rotation,
     input wire signed [17:0] cosfactor, sinfactor);

    parameter grey = 0;
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    parameter xcubecenter = (1024/2);
    parameter ycubecenter = (768/2);

    reg signed [11:0] xmatrixA, xmatrixB, xmatrixC, xmatrixD, xmatrixE, xprimeAsigned;

    reg signed [10:0] ymatrixA, ymatrixB, ymatrixC, ymatrixD, ymatrixE, yprimeAsigned;

    reg signed [10:0] zmatrixA, zmatrixB, zmatrixC, zmatrixD, zmatrixE, zmatrixF, zprimeAsigned;

    reg [2:0] colouroutp1, colouroutp2, colouroutp3, colouroutp4, colouroutp5;
    reg control_bitp1, control_bitp2, control_bitp3, control_bitp4, control_bitp5;


   always @ * begin
      if (((hcount >= (x + border_thickness)) && (hcount < (x+WIDTH+border_thickness))) && ((
      vcount >= (y + border_thickness)) && (vcount < (y+WIDTH + border_thickness))) && (dcount
      == z)) colouroutp1 = colour;
        //else colourout = black;
        else colouroutp1 = black;

      if (((hcount >= (x+1)) && (hcount < (x+WIDTH))) && ((vcount >= (y+1)) && (vcount < (y+
      WIDTH))) && (dcount == z)) control_bitp1 = 1;
        else control_bitp1 = 0;


   end

    always @(posedge vclock) begin

        case(rotation)

        2: begin // rotation about z axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17); //
            /$unsigned(256));
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA + ymatrixB + 150;
            ymatrixC <= - xmatrixB + ymatrixA + 150;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        1:  begin //rotation about y axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            ymatrixA <= $unsigned(vcount); // /$unsigned(256));
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17);
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17);
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA - zmatrixB + 150;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA + xmatrixB +150;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        3: begin // rotation about x axis

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17); //
            /$unsigned(256));
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA + zmatrixB + 150;
            zmatrixC <= - ymatrixB + zmatrixA + 150;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;
```

```
            end

        0: begin //no rotation

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        default: begin //no rotation

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        endcase

        xmatrixD <= (xmatrixC*113512)>>>17; // /($unsigned(256));
        ymatrixD <= (zmatrixC*113512)>>>17; // /($unsigned(256));
        xmatrixE <= (xmatrixC)>>>1; // /$unsigned(2);
        ymatrixE <= (zmatrixC)>>>1; // /$unsigned(2);
        zmatrixD <= (xmatrixC*92682)>>>17; // /($unsigned(256));
        zmatrixE <= (zmatrixC*92682)>>>17; // /($unsigned(256));
        zmatrixF <= ymatrixC;
        colouroutp4 <= colouroutp3;
        control_bitp4 <= control_bitp3;
        xprimeAsigned <= xcubecenter + ymatrixD - xmatrixD;
        yprimeAsigned <= zmatrixF + ycubecenter - xmatrixE - ymatrixE;
        zprimeAsigned <= zmatrixE + zmatrixD + 86;
        colouroutp5 <= colouroutp4;
        control_bitp5 <= control_bitp4;

        xprime<= xprimeAsigned[10:0];
        yprime<= yprimeAsigned[10:0];
        zprime<= zprimeAsigned[10:0];
        colourout <= colouroutp5;
        control_bit <= control_bitp5;

    end


endmodule

//almost identical to top_face => same comments!
module right_face
   #(parameter WIDTH = 90,            // default width: 100 pixels
               border_thickness = 5)
   (input wire vclock,
     input wire [10:0] x,hcount,
    input wire[9:0] y,vcount,
     input wire[9:0] z,dcount,
    input wire [2:0] colour,
     output reg [2:0] colourout,
     output reg [10:0] xprime,
     output reg [9:0] yprime,
     output reg [9:0] zprime,
     output reg control_bit,
     input wire [1:0] rotation,
     input wire signed [17:0] cosfactor, sinfactor);

    parameter grey = 0;
    parameter red = 1;
    parameter orange = 2;
    parameter yellow = 3;
    parameter green = 4;
    parameter blue = 5;
    parameter white = 6;
    parameter black = 7;

    parameter xcubecenter = (1024/2);
    parameter ycubecenter = (768/2);

    reg signed [11:0] xmatrixA, xmatrixB, xmatrixC, xmatrixD, xmatrixE, xprimeAsigned;

    reg signed [10:0] ymatrixA, ymatrixB, ymatrixC, ymatrixD, ymatrixE, yprimeAsigned;

    reg signed [10:0] zmatrixA, zmatrixB, zmatrixC, zmatrixD, zmatrixE, zmatrixF, zprimeAsigned;

    reg [2:0] colouroutp1, colouroutp2, colouroutp3, colouroutp4, colouroutp5;
    reg control_bitp1, control_bitp2, control_bitp3, control_bitp4, control_bitp5;


   always @ * begin
      if (((hcount >= (x + border_thickness)) && (hcount < (x+WIDTH+border_thickness))) && ((
      vcount >= (y + border_thickness)) && (vcount < (y+WIDTH + border_thickness))) && (dcount
      == z)) colouroutp1 = colour;
        else colouroutp1 = black;

      if (((hcount >= (x+1)) && (hcount < (x+WIDTH))) && ((vcount >= (y+1)) && (vcount < (y+
      WIDTH))) && (dcount == z)) control_bitp1 = 1;
        else control_bitp1 = 0;

    //colourout = colouroutp3;

   end
    always @(posedge vclock) begin

        case(rotation)

        3: begin // rotation about z axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17); //
            /$unsigned(256));
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA - ymatrixB + 150;
            ymatrixC <= xmatrixB + ymatrixA + 150;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        1:  begin //rotation about y axis

            xmatrixA <= ((($unsigned(hcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            xmatrixB <= ((($unsigned(hcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            ymatrixA <= $unsigned(vcount); // /$unsigned(256));
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17);
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17);
```

```
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA + zmatrixB + 150;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA - xmatrixB +150;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        2: begin // rotation about x axis

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= ((($unsigned(vcount)-$unsigned(150))*cosfactor)>>>17); //
            /$unsigned(256));
            ymatrixB <= ((($unsigned(vcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            zmatrixA <= ((($unsigned(dcount)-$unsigned(150))*cosfactor)>>>17); //
            /($unsigned(256)));
            zmatrixB <= ((($unsigned(dcount)-$unsigned(150))*sinfactor)>>>17); //
            /$unsigned(2)));
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA - zmatrixB + 150;
            zmatrixC <= ymatrixB + zmatrixA + 150;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        0: begin //no rotation

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end
        default: begin //no rotation

            xmatrixA <= $unsigned(hcount);
            xmatrixB <= 0;
            ymatrixA <= $unsigned(vcount);
            ymatrixB <= 0; // /$unsigned(2)));
            zmatrixA <= $unsigned(dcount);
            zmatrixB <= 0;
            colouroutp2 <= colouroutp1;
            control_bitp2 <= control_bitp1;

            xmatrixC <= xmatrixA;
            ymatrixC <= ymatrixA;
            zmatrixC <= zmatrixA;
            colouroutp3 <= colouroutp2;
            control_bitp3 <= control_bitp2;

            end

        endcase

        xmatrixD <= (zmatrixC*113512)>>>17; // /($unsigned(256));
        ymatrixD <= (xmatrixC*113512)>>>17; // /($unsigned(256));
        xmatrixE <= (zmatrixC)>>>1; // /$unsigned(2);
        ymatrixE <= (xmatrixC)>>>1; // /$unsigned(2);
        zmatrixD <= (zmatrixC*92682)>>>17; // /($unsigned(256));
        zmatrixE <= (xmatrixC*92682)>>>17; // /($unsigned(256));
        zmatrixF <= ymatrixC;
        colouroutp4 <= colouroutp3;
        control_bitp4 <= control_bitp3;

        xprimeAsigned <= xcubecenter + ymatrixD - xmatrixD;
        yprimeAsigned <= zmatrixF + ycubecenter - xmatrixE - ymatrixE;
        zprimeAsigned <= zmatrixE + zmatrixD + 86;
        colouroutp5 <= colouroutp4;
        control_bitp5 <= control_bitp4;

        xprime<= xprimeAsigned[10:0];
        yprime<= yprimeAsigned[10:0];
        zprime<= zprimeAsigned[10:0];
        colourout <= colouroutp5;
        control_bit <= control_bitp5;

    end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Jack
//////////////////////////////////////////////////////////////////////////////////

module rotator(
    input wire vclock,
    input wire start,
    input wire[4:0] rotation,
    output reg[1:0] rott1,rott2,rott3,rott4,rott5,rott6,rott7,rott8,rott9,  //rotation type to
    be performed for each top face
    output reg[1:0] rotr1,rotr2,rotr3,rotr4,rotr5,rotr6,rotr7,rotr8,rotr9,  //rotation type to
    be performed for each right face
    output reg[1:0] rotl1,rotl2,rotl3,rotl4,rotl5,rotl6,rotl7,rotl8,rotl9,  //rotation type to
    be performed for each left face
    output reg signed[17:0] cosfactor, sinfactor,
    output reg complete,
    input wire [0:161] cube_config,                                         //cube config from
    Solve Cube Block
    output reg [ 0:161] displayed_cube_config                               //cube config
    currently on screen
    );

    parameter idle = 0;
    parameter rotating = 1;

    reg state1, rotateclk;
    reg [4:0]pos;
    reg signed[17:0] sinfactorprime;
    reg [21:0] counter;

    always @ (posedge rotateclk) begin
        if (state1 == rotating) begin
            if (pos < 5'd30) begin
                pos <= pos+1;
                //complete <= 0;
            end
            else begin
                pos <=0;
                //complete <= 1;
            end
        end
        else begin
            pos <= 0;
            //complete <= 0;
        end
    end

    always @ (posedge vclock) begin

        if(counter == 2_150_000) begin // create 15 Hz clock for 15 fps animation
            counter <= 0;
            rotateclk <= ~rotateclk;
            end
        else counter <= counter+1;

        case(state1)

            idle: begin //wait for user to initialise next rotation

                        if(start) begin
                            state1 <= rotating;
                        end
                        else begin

                            rott1<=0;
                            rott2<=0;
                            rott3<=0;
                            rott4<=0;
                            rott5<=0;
                            rott6<=0;
                            rott7<=0;
                            rott8<=0;
                            rott9<=0;

                            rotl1<=0;
                            rotl2<=0;
                            rotl3<=0;
                            rotl4<=0;
                            rotl5<=0;
                            rotl6<=0;
                            rotl7<=0;
                            rotl8<=0;
                            rotl9<=0;

                            rotr1<=0;
                            rotr2<=0;
                            rotr3<=0;
                            rotr4<=0;
                            rotr5<=0;
                            rotr6<=0;
                            rotr7<=0;
                            rotr8<=0;
                            rotr9<=0;

                        end
                    end

            rotating: begin

                    if(pos >= 30) begin //wait for the rotation animation to complete all 30
                    frames
                        state1 <= idle;
                        complete <= 1;
                    end
                    else complete <= 0;
```

```
                case(rotation) //set the cubelets to be rotated and which rotation to do
                based on the rotation type received from the solve cube clock
                    //sinefactorprime is negative for reversed rotations
                    0:  begin

                            rott1<=0;
                            rott2<=0;
                            rott3<=0;
                            rott4<=0;
                            rott5<=0;
                            rott6<=0;
                            rott7<=0;
                            rott8<=0;
                            rott9<=0;

                            rotl1<=0;
                            rotl2<=0;
                            rotl3<=0;
                            rotl4<=0;
                            rotl5<=0;
                            rotl6<=0;
                            rotl7<=0;
                            rotl8<=0;
                            rotl9<=0;

                            rotr1<=0;
                            rotr2<=0;
                            rotr3<=0;
                            rotr4<=0;
                            rotr5<=0;
                            rotr6<=0;
                            rotr7<=0;
                            rotr8<=0;
                            rotr9<=0;

                            sinfactor <= sinfactorprime;

                        end

                    1:  begin

                            rott1<=3;
                            rott2<=0;
                            rott3<=0;
                            rott4<=3;
                            rott5<=0;
                            rott6<=0;
                            rott7<=3;
                            rott8<=0;
                            rott9<=0;

                            rotl1<=3;
                            rotl2<=0;
                            rotl3<=0;
                            rotl4<=3;
                            rotl5<=0;
                            rotl6<=0;
                            rotl7<=3;
                            rotl8<=0;
                            rotl9<=0;

                            rotr1<=3;
                            rotr2<=3;
                            rotr3<=3;
                            rotr4<=3;
                            rotr5<=3;
                            rotr6<=3;
                            rotr7<=3;
                            rotr8<=3;
                            rotr9<=3;

                            sinfactor <= sinfactorprime;
                        end

                    2:  begin

                            rott1<=3;
                            rott2<=0;
                            rott3<=0;
                            rott4<=3;
                            rott5<=0;
                            rott6<=0;
                            rott7<=3;
                            rott8<=0;
                            rott9<=0;

                            rotl1<=3;
                            rotl2<=0;
                            rotl3<=0;
                            rotl4<=3;
                            rotl5<=0;
                            rotl6<=0;
                            rotl7<=3;
                            rotl8<=0;
                            rotl9<=0;

                            rotr1<=3;
                            rotr2<=3;
                            rotr3<=3;
                            rotr4<=3;
                            rotr5<=3;
                            rotr6<=3;
                            rotr7<=3;
                            rotr8<=3;
                            rotr9<=3;
```

```
                        sinfactor <= -sinfactorprime;
                    end

                3:  begin

                        rott1<=0;
                        rott2<=3;
                        rott3<=0;
                        rott4<=0;
                        rott5<=3;
                        rott6<=0;
                        rott7<=0;
                        rott8<=3;
                        rott9<=0;

                        rotl1<=0;
                        rotl2<=3;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=3;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=3;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= sinfactorprime;
                    end

                4:  begin

                        rott1<=0;
                        rott2<=3;
                        rott3<=0;
                        rott4<=0;
                        rott5<=3;
                        rott6<=0;
                        rott7<=0;
                        rott8<=3;
                        rott9<=0;

                        rotl1<=0;
                        rotl2<=3;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=3;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=3;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= -sinfactorprime;
                    end

                5:  begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=3;
                        rott4<=0;
                        rott5<=0;
                        rott6<=3;
                        rott7<=0;
                        rott8<=0;
                        rott9<=3;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=3;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=3;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=3;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;
                        sinfactor <= sinfactorprime;
                    end

                6:  begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=3;
                        rott4<=0;
                        rott5<=0;
                        rott6<=3;
                        rott7<=0;
                        rott8<=0;
                        rott9<=3;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=3;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=3;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=3;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= -sinfactorprime;
                    end

                7:  begin

                        rott1<=2;
                        rott2<=2;
                        rott3<=2;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=0;
                        rott8<=0;
                        rott9<=0;

                        rotl1<=2;
                        rotl2<=2;
                        rotl3<=2;
```

```
                    rotl4<=2;
                    rotl5<=2;
                    rotl6<=2;
                    rotl7<=2;
                    rotl8<=2;
                    rotl9<=2;

                    rotr1<=2;
                    rotr2<=0;
                    rotr3<=0;
                    rotr4<=2;
                    rotr5<=0;
                    rotr6<=0;
                    rotr7<=2;
                    rotr8<=0;
                    rotr9<=0;

                    sinfactor <= sinfactorprime;

                end

            8:  begin

                    rott1<=2;
                    rott2<=2;
                    rott3<=2;
                    rott4<=0;
                    rott5<=0;
                    rott6<=0;
                    rott7<=0;
                    rott8<=0;
                    rott9<=0;

                    rotl1<=2;
                    rotl2<=2;
                    rotl3<=2;
                    rotl4<=2;
                    rotl5<=2;
                    rotl6<=2;
                    rotl7<=2;
                    rotl8<=2;
                    rotl9<=2;

                    rotr1<=2;
                    rotr2<=0;
                    rotr3<=0;
                    rotr4<=2;
                    rotr5<=0;
                    rotr6<=0;
                    rotr7<=2;
                    rotr8<=0;
                    rotr9<=0;
                    sinfactor <= - sinfactorprime;

                end

            9:  begin

                    rott1<=0;
                    rott2<=0;
                    rott3<=0;
                    rott4<=2;
                    rott5<=2;
                    rott6<=2;
                    rott7<=0;
                    rott8<=0;
                    rott9<=0;

                    rotl1<=0;
                    rotl2<=0;
                    rotl3<=0;
                    rotl4<=0;
                    rotl5<=0;
                    rotl6<=0;
                    rotl7<=0;
                    rotl8<=0;
                    rotl9<=0;

                    rotr1<=0;
                    rotr2<=2;
                    rotr3<=0;
                    rotr4<=0;
                    rotr5<=2;
                    rotr6<=0;
                    rotr7<=0;
                    rotr8<=2;
                    rotr9<=0;

                    sinfactor <= sinfactorprime;

                end

            10: begin

                    rott1<=0;
                    rott2<=0;
                    rott3<=0;
                    rott4<=2;
                    rott5<=2;
                    rott6<=2;
                    rott7<=0;
                    rott8<=0;
                    rott9<=0;

                    rotl1<=0;
```

```
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=2;
                        rotr3<=0;
                        rotr4<=0;
                        rotr5<=2;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=2;
                        rotr9<=0;

                        sinfactor <= - sinfactorprime;

                    end

                11: begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=0;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=2;
                        rott8<=2;
                        rott9<=2;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=2;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=2;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=2;

                        sinfactor <= sinfactorprime;

                    end

                12: begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=0;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=2;
                        rott8<=2;
                        rott9<=2;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=2;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=2;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=2;

                        sinfactor <= - sinfactorprime;

                    end

                13: begin

                        rott1<=1;
                        rott2<=1;
                        rott3<=1;
                        rott4<=1;
                        rott5<=1;
                        rott6<=1;
                        rott7<=1;
                        rott8<=1;
                        rott9<=1;
                        rotl1<=1;
                        rotl2<=1;
                        rotl3<=1;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=1;
                        rotr2<=1;
                        rotr3<=1;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= -sinfactorprime;

                    end

                14: begin

                        rott1<=1;
                        rott2<=1;
                        rott3<=1;
                        rott4<=1;
                        rott5<=1;
                        rott6<=1;
                        rott7<=1;
                        rott8<=1;
                        rott9<=1;

                        rotl1<=1;
                        rotl2<=1;
                        rotl3<=1;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=1;
                        rotr2<=1;
                        rotr3<=1;
                        rotr4<=0;
                        rotr5<=0;
                        rotr6<=0;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= sinfactorprime;

                    end

                15: begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=0;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=0;
                        rott8<=0;
                        rott9<=0;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=1;
                        rotl5<=1;
                        rotl6<=1;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=1;
                        rotr5<=1;
                        rotr6<=1;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= -sinfactorprime;

                    end

                16: begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=0;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=0;
                        rott8<=0;
                        rott9<=0;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=1;
                        rotl5<=1;
                        rotl6<=1;
                        rotl7<=0;
                        rotl8<=0;
                        rotl9<=0;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=1;
                        rotr5<=1;
                        rotr6<=1;
                        rotr7<=0;
                        rotr8<=0;
                        rotr9<=0;

                        sinfactor <= sinfactorprime;

                    end

                17: begin

                        rott1<=0;
                        rott2<=0;
                        rott3<=0;
                        rott4<=0;
                        rott5<=0;
                        rott6<=0;
                        rott7<=0;
                        rott8<=0;
                        rott9<=0;

                        rotl1<=0;
                        rotl2<=0;
                        rotl3<=0;
                        rotl4<=0;
                        rotl5<=0;
                        rotl6<=0;
                        rotl7<=1;
                        rotl8<=1;
                        rotl9<=1;

                        rotr1<=0;
                        rotr2<=0;
                        rotr3<=0;
                        rotr4<=0;
```

```
                    rotr5<=0;
                    rotr6<=0;
                    rotr7<=1;
                    rotr8<=1;
                    rotr9<=1;

                    sinfactor <= -sinfactorprime;

                end

            18: begin

                    rott1<=0;
                    rott2<=0;
                    rott3<=0;
                    rott4<=0;
                    rott5<=0;
                    rott6<=0;
                    rott7<=0;
                    rott8<=0;
                    rott9<=0;

                    rotl1<=0;
                    rotl2<=0;
                    rotl3<=0;
                    rotl4<=0;
                    rotl5<=0;
                    rotl6<=0;
                    rotl7<=1;
                    rotl8<=1;
                    rotl9<=1;

                    rotr1<=0;
                    rotr2<=0;
                    rotr3<=0;
                    rotr4<=0;
                    rotr5<=0;
                    rotr6<=0;
                    rotr7<=1;
                    rotr8<=1;
                    rotr9<=1;

                    sinfactor <= sinfactorprime;

                end

                default :begin

                    rott1<=0;
                    rott2<=0;
                    rott3<=0;
                    rott4<=0;
                    rott5<=0;
```

```
                                rott6<=0;
                                rott7<=0;
                                rott8<=0;
                                rott9<=0;

                                rotl1<=0;
                                rotl2<=0;
                                rotl3<=0;
                                rotl4<=0;
                                rotl5<=0;
                                rotl6<=0;
                                rotl7<=0;
                                rotl8<=0;
                                rotl9<=0;

                                rotr1<=0;
                                rotr2<=0;
                                rotr3<=0;
                                rotr4<=0;
                                rotr5<=0;
                                rotr6<=0;
                                rotr7<=0;
                                rotr8<=0;
                                rotr9<=0;

                                sinfactor <= sinfactorprime;

                            end

                    endcase
                end
            default: state1 <= 0;

        endcase
    end

   //send the correct sine and cosine value to the face module based on the current rotation
   angle
   //increment the angle by 3 degrees every 1/15 of a second
    always @(*) begin

        case(pos)

            0:  begin //0 degrees
                    sinfactorprime = 0;
                    cosfactor = 131071;
                end

            1:  begin //3 degrees
                    sinfactorprime = 6860;
                    cosfactor = 130892;
                end
            2:  begin //6 degrees
                    sinfactorprime = 13700;
                    cosfactor = 130353;
                end

            3:  begin //9 degrees
                    sinfactorprime = 20504;
                    cosfactor = 129458;
                end

            4:  begin //12 degrees
                    sinfactorprime = 27251;
                    cosfactor = 128207;
                end

            5:  begin //15 degrees
                    sinfactorprime = 33923;
                    cosfactor = 126605;
                end

            6:  begin //18 degrees
                    sinfactorprime = 40503;
                    cosfactor = 124656;
                end

            7:  begin //21 degrees
                    sinfactorprime = 46972;
                    cosfactor = 122366;
                end

            8:  begin //24 degrees
                    sinfactorprime = 53311;
                    cosfactor = 119740;
                end

            9:  begin //27 degrees
                    sinfactorprime = 59505;
                    cosfactor = 116786;
                end

            10:begin //30 degrees
                    sinfactorprime = 65536;
                    cosfactor = 113511;
                end

            11:begin //33 degrees
                    sinfactorprime = 71386;
                    cosfactor = 109926;
                end

            12:begin //36 degrees
                    sinfactorprime = 77042;
                    cosfactor = 106039;
```

```
        end

    13:begin //39 degrees
            sinfactorprime = 82486;
            cosfactor = 101862;
        end

    14:begin //42 degrees
            sinfactorprime = 87704;
            cosfactor = 97405;
        end

    15:begin //-45 degrees and switch to next configuration (should look continuous on
    screen)
            sinfactorprime = -92681;
            cosfactor = 92681;
            displayed_cube_config <= cube_config;
        end

    16:begin //-42 degrees
            sinfactorprime = -87704;
            cosfactor = 97405;
        end

    17:begin //-39 degrees
            sinfactorprime = -82486;
            cosfactor = 101862;
        end

    18:begin //-36 degrees
            sinfactorprime = -77042;
            cosfactor = 106039;
        end

    19:begin //-33 degrees
            sinfactorprime = -71386;
            cosfactor = 109926;
        end

    20:begin //-30 degrees
            sinfactorprime = -65536;
            cosfactor = 113511;
        end

    21:begin //-27 degrees
            sinfactorprime = -59505;
            cosfactor = 116786;
        end

    22:begin //-24 degrees
            sinfactorprime = -53311;
            cosfactor = 119740;
        end
```

```
            23:begin //-21 degrees
                    sinfactorprime = -46972;
                    cosfactor = 122366;
                end

            24:begin //-18 degrees
                    sinfactorprime = -40503;
                    cosfactor = 124656;
                end

            25:begin //-15 degrees
                    sinfactorprime = -33923;
                    cosfactor = 126605;
                end

            26:begin //-12 degrees
                    sinfactorprime = -27251;
                    cosfactor = 128207;
                end

            27:begin //-9 degrees
                    sinfactorprime = -20504;
                    cosfactor = 129458;
                end

            28:begin //-6 degrees
                    sinfactorprime = -13700;
                    cosfactor = 130353;
                end

            29:begin //-3 degrees
                    sinfactorprime = -6860;
                    cosfactor = 130892;
                end

            default: begin
                    sinfactorprime = 0;
                    cosfactor = 131071;
                end
        endcase
    end

endmodule
```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Author: Jack
////////////////////////////////////////////////////////////////////////////////

module serial_receive
    #(parameter config_length=162) //by default set to length for 54 cubelet face cube config

    (
    input external_clk,     //clock from other FPGA ~100kHz
    input sync_pulse,
    input data,             //serial data in
    output reg[4:0] rotation,
    output reg[0:config_length -1] cube_config
    );

    reg [config_length + 5 - 1:0] data_received; //shift register to receive data
    reg [7:0] count;
    reg state;

    always @ (negedge external_clk) begin

        case(state)

        0: begin //wait for data to be sent
                if(sync_pulse) begin //upon sync pulse start shifting in data
                    count<=0;
                    state <= 1;
                end
            end

        1: begin
                if(count < (config_length + 5)) begin   //shift in data until all data is
                received
                    data_received [0] <= data;
                    data_received [config_length + 5 - 1:1] <= data_received[config_length + 5 -
                     2:0];
                    count <= count + 1;
                end

                else begin //once data received update cube_config and rotation to new values
                    rotation <= data_received[4:0];
                    cube_config[0:config_length - 1] <= data_received [config_length + 5 - 1:5];
                    state <= 0;
                end
            end

        default: state <= 0;

        endcase
    end

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
// Author: 6.111 Course Staff
// Description: Debounces a signal to create a clean signal
///////////////////////////////////////////////////////////////////////////////

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

   reg [19:0] count;
   reg new;

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

endmodule
```

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- 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
```

```
###############################################################################
#
# 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";

#
# 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;
```