Termanator 3D

Robert Cassels and Tim Mamtora

6.111 Introductory Digital Systems Laboratory

December 9, 2004

Abstract

This report describes the implementation in Verilog of a first person 3D Video game using the technique of ray casting. The goal of the game is to negotiate a 3D maze past opponents to the end of the level. The user interface is a keyboard controller with a VGA display on screen. The internal design consists of numerous blocks, a 3D Renderer to formulate the image on screen, positioning control for the player and game characters as well as an AI block to control opponent movements and simple game logic to signal and display constant items on screen. The keyboard and VGA interfaces were tested separately and first, before building in 3D elements and finally incorporating the game logic. The final outcome was a pleasing success with working user interfaces and a 3D image on screen minus the AI characters.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Game Overview

The aim of the project is to produce a system that will allow a user to navigate a three dimensional (3D) maze. The player's aim is to travel from the start to the end of the maze.

The player's progress through the maze will be impeded by the presence of artificially intelligent characters (AICs). An AIC may attack the player only when in direct contact with the player. The player may defend themselves using a gun. (All items in the previous sentence refer to virtual events and items.)

Attacks on the player by the AICs will result in the loss of some of the player's health points. Similarly, if the player shoots an AIC, then the AIC will lose health points. If an AICs health level reach zero, it will be removed from the game (called a death event). If the player's health points reach zero, then the game terminates with a player loss result.

The player wins by reaching the maze exit.

1.2 Basic Architecture

The system must accept valid player requests via a keyboard and act accordingly in the game environment. At the same time the movement of AICs must be coordinated and all of this information must be displayed on screen. The basic top-level architecture is shown in Figure 1.

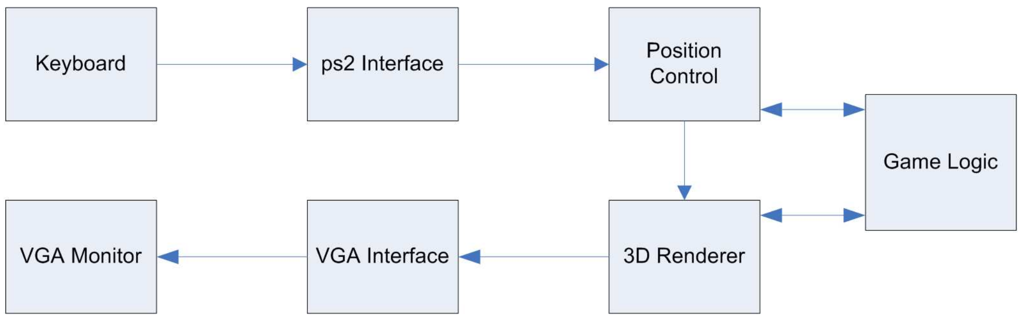

Figure 1: Basic Block Diagram of the Overall System

There are two items used for the User Interface (UI). These are the VGA Monitor for video output and the PS/2 Keyboard for user input. For each peripheral device (monitor/keyboard) there exists an interface block in the system. This converts required information received externally or generated internally to a useful format for the relevant destination e.g. the keyboard serially sends a unique scan code per key pressed and this must be converted into a position request by the player for the rest of the system to interpret.

The 3D Renderer generates data for output to the Monitor by the Video Interface. The Position Control block must retain the current character positions as well as updating the player position based upon the data retrieved from the keyboard. The Game Logic must provide coordination for the other modules as well as providing the higher level representation of the game rules.

1.3 Background Information

1.3.1 Ray Casting

Ray Casting is a simplification of a 3D Rendering technique called Ray-Tracing. Ray-Casting works by "shooting" rays from the camera (your eyes) position and finding out which surface they hit first. This surface will then provide the colour, which can be displayed at the pixel through which the ray was fired.

Several constraints imposed on this scheme can improve the amount of computation needed substantially.

- Walls and floors/ceilings must be perpendicular.
- Walls are aligned to a grid.
- Floors and ceilings are flat.

This means that only intersections of grids need to be checked to see if a wall is present. Since the gradient of a line is constant, incremental values can simply be added to find the end position of the ray. There are two increments, one for walls aligned with one coordinate axis and another for those in the other. The first positions that are calculated are those closest to the player. To get these, the distance to the nearest boundary in each direction is found and the the following equations are used to caculate their corresponding distances along the other axis.

$$change_y = \frac{bounday_x \times increment_y}{BlockWidth} = \frac{bounday_x \times increment_y}{64}$$
$$change_x = \frac{bounday_y \times increment_x}{BlockWidth} = \frac{bounday_y \times increment_x}{64}$$

These values are then incremented and the one nearest the player chosen until a wall has been found.

The AIC's corresponding information is calculated by forming the difference between the palyer's and the AIC's positions in both the X and Y directions. Two operations are performed with these values (which we shall name dx and dy). The first operations is $d = dx^2 + dy^2$, where d is the distance and the second is $\theta = \arctan(\frac{dy}{dx})$

The distances from these are used to calculate intensity, height on screen and height scaling values so that the texture coordinates can be derived from the screen X and Y positions.

1.3.2 VGA Monitor

The VGA system uses two synchronisation channels, horizontal and vertical together with three analogue colour wires for the principal colours red, green and blue.

1.3.3 Hardware Resources

The system is implemented in hardware using a Xilinx Spartan 3 FPGA Starter Kit. The video output is controlled via an Analogue Devices ADV7120 video Digital to Analogue Converter (DAC) to achieve 24-bit colour.

2 Module Description and Implementation

2.1 Detailed Architecture

A more detailed block diagram of the system as a whole is shown in Figure 2.

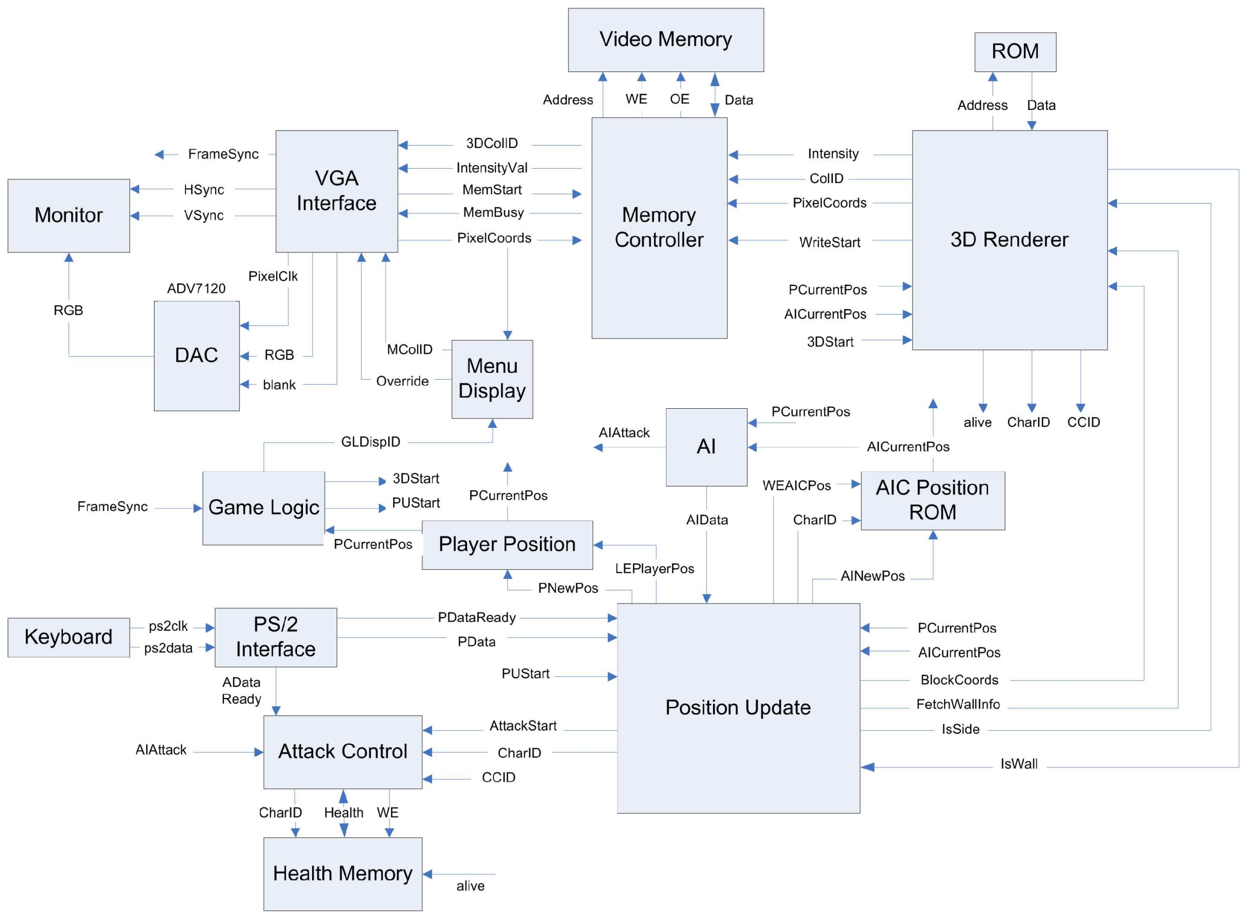

Figure 2: Detailed Block Diagram of the Overall System

The principle units of this were designed separately with overall control handed to a hierarchy of Finite State Machines (FSMs). Each module has reference to a global clock and a reset signal to ensure simultaneous timing among all blocks. The inputs and outputs from the overall system concern only the user interfaces for the player. These are shown in Table 1.

2.2 VGA Interface (TM)

The function of the VGA interface is to output the valid pixel data to the screen, interfacing with the ADV7120 Video DAC and the monitor itself. A block diagram of the system is shown in Figure 3.

The outputs to the DACs require a Pixel Clk (whose rate is governed by the required resolution) in addition to three 8-bit colour lines (red, green and blue) for a 24-bit spectrum of colours. A blanking signal provided also blanks the screen regardless of colours supplied, required during “flyback” to reset the pixel coordinates for the next frame.

Table 1: System Input/Output Specifications

Signal Name	Input/Output	Signal Type	Signal Description
HS	Output	Digital	Horizontal Synchronisation for VGA line output
VS	Output	Digital	Vertical Synchronisation for VGA frame output
R	Output	Analogue Voltage Level	Red Pixel Line for VGA output
G	Output	Analogue Voltage Level	Green Pixel Line for VGA output
B	Output	Analogue Voltage Level	Blue Pixel Line for VGA output
Blank	Output	Digital	Blanking signal to DACs for VGA "flyback"
ps2Clk	Input	Digital	Serial Clk Synchronisation for PS/2 interface
ps2Data	Input	Digital	Serial Data input from Keyboard

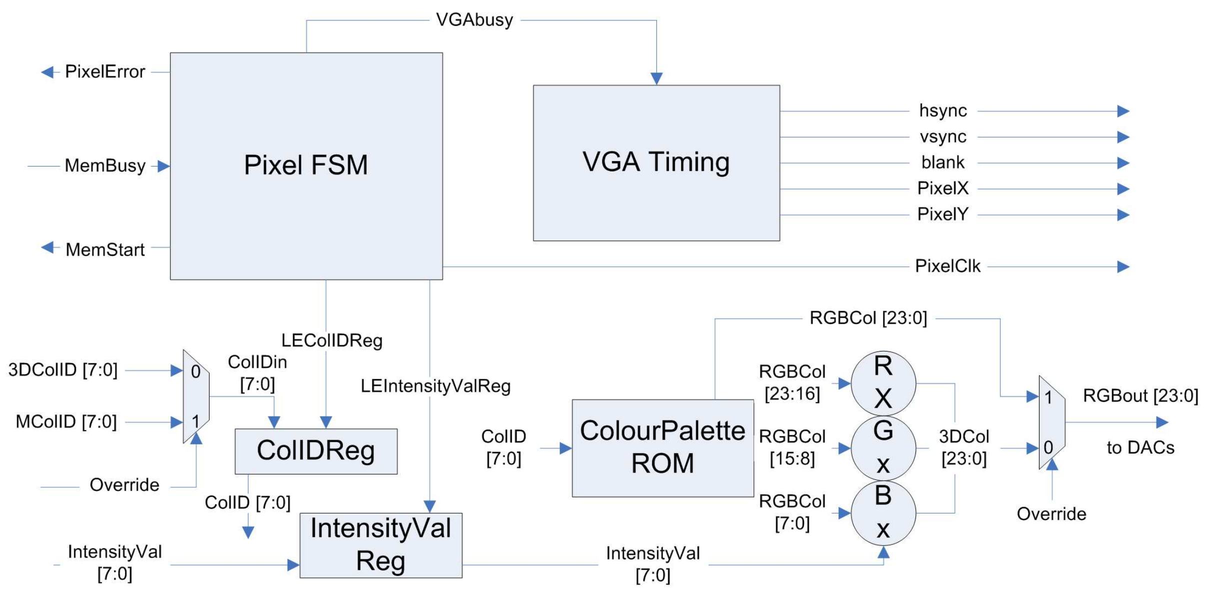

Figure 3: A Block Diagram of the VGA Interface

The video interface must also time the horizontal sync and vertical sync signals to ensure a correct picture is displayed on screen. This is done via two pixel counts (x and y), supplying a horizontal sync pulse (active low) at multiples of the vertical resolution and vertical sync pulses (active low) at multiples of the frame rate. A 320x 240 resolution was used and since the lowest resolution on the monitor was 640x480 the counts were run as normal but displaying the horizontal lines twice for the vertical resolution. For a normal clock rate of 25MHz for a 640x480 display this meant that a pixel clock of 12.5Mhz was required for the timing (and for the DAC). This was achieved using a pulse every four clock cycles governed by Pixel FSM (since this is being run at the system clock rate of 50Mhz.

The operation of the whole interface is as follows. The VGA Timing supplies an x and y pixel at each count and this is used to lookup the next colour to be output from the video memory (via the memory controller). The Pixel FSM will initiate a read every four clock cycles, during which time it will wait until ready and then load the ColID and IntensityVal values returned into their respective registers. This will then propagate through the ROM lookup and the multiplier with enough time to reach the output to the DACs whilst the timing module supplies the next pixel to be looked up. Upon the Pixel FSM reaching idle the Pixel Clk is pulsed high and VGA output enabled to output the relevant pixel to the screen and repeat the process again. Pixel FSM has only 4 states in one cycle to run through which it does constantly, off which the VGA timing is synchronised. Figure 4 shows the state transition diagram for the Pixel FSM.

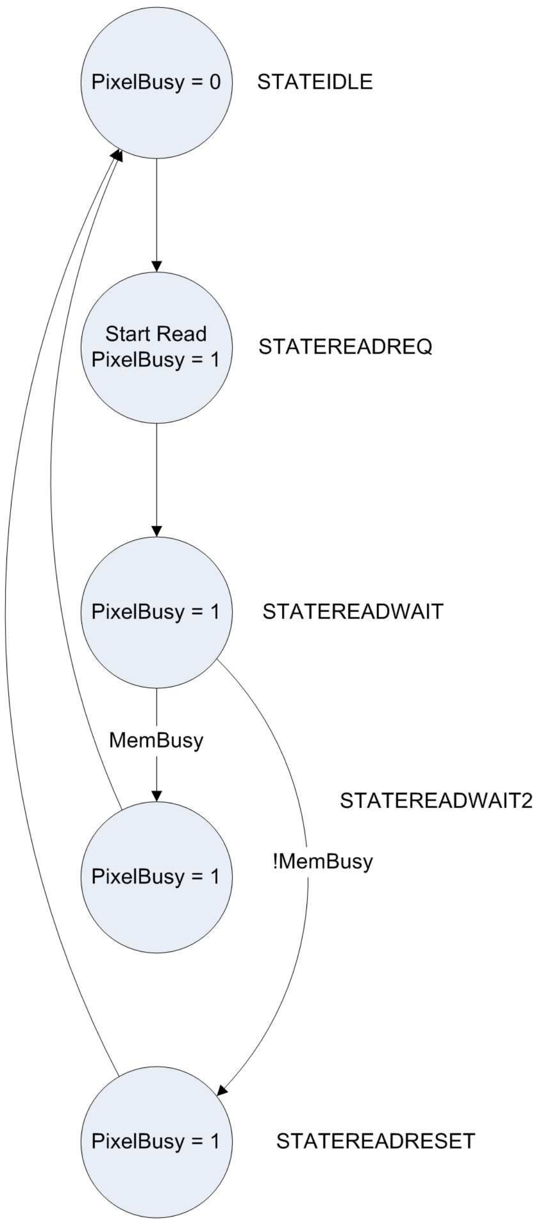

Figure 4: A State Transition Diagram for Pixel FSM module

2.3 Video Memory Access Controller (RC)

This controller handles the interface to the Video RAM. It allows two modules to read and write in a timely fashion without causing contention on the address and data busses. The state transition diagram can be seen in Figure 5.

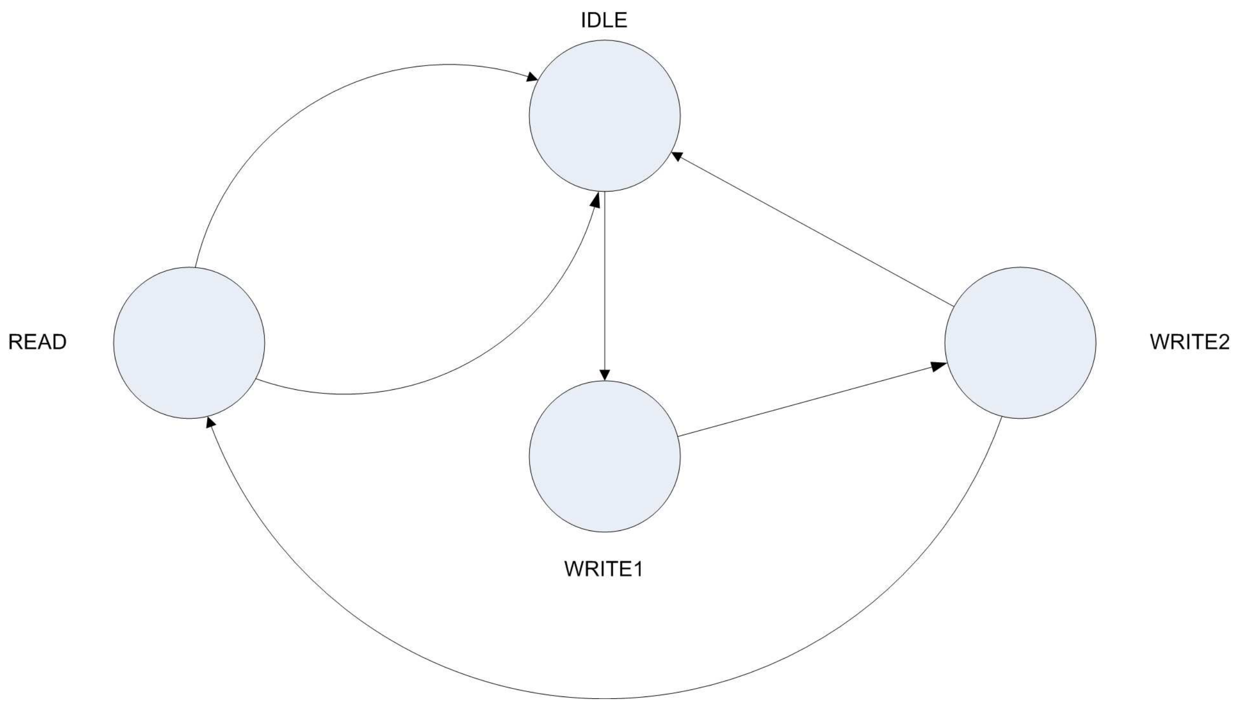

Figure 5: A State Transition Diagram for Video Memory Access

2.4 3D Renderer

The 3D Renderer is responsible for producing values indicating the colour and intensity of every pixel in a single frame as seen by the *camera* (the camera is a term describing the point of view from which a 3D scene is rendered). The information with which these values are calculated is the current player position and orientation (which are the same as for the camera in this case), the positions of the AICs, and the placement of walls on the map. The background for creation of these values can be found in Section 1.3.1.

This section describes the implementation of these techniques and disusses the limitations and constraints imposed practically. The overall block diagram for the 3D Renderer can be seen in Figure 6

2.4.1 ROM Controller

The ROM Controller deals with the actual accessing of the ROM to retrieve several lookup values for different purposes. It also provides storage in the form of registers for many of these values. The values it looks up are listed in Table 2.

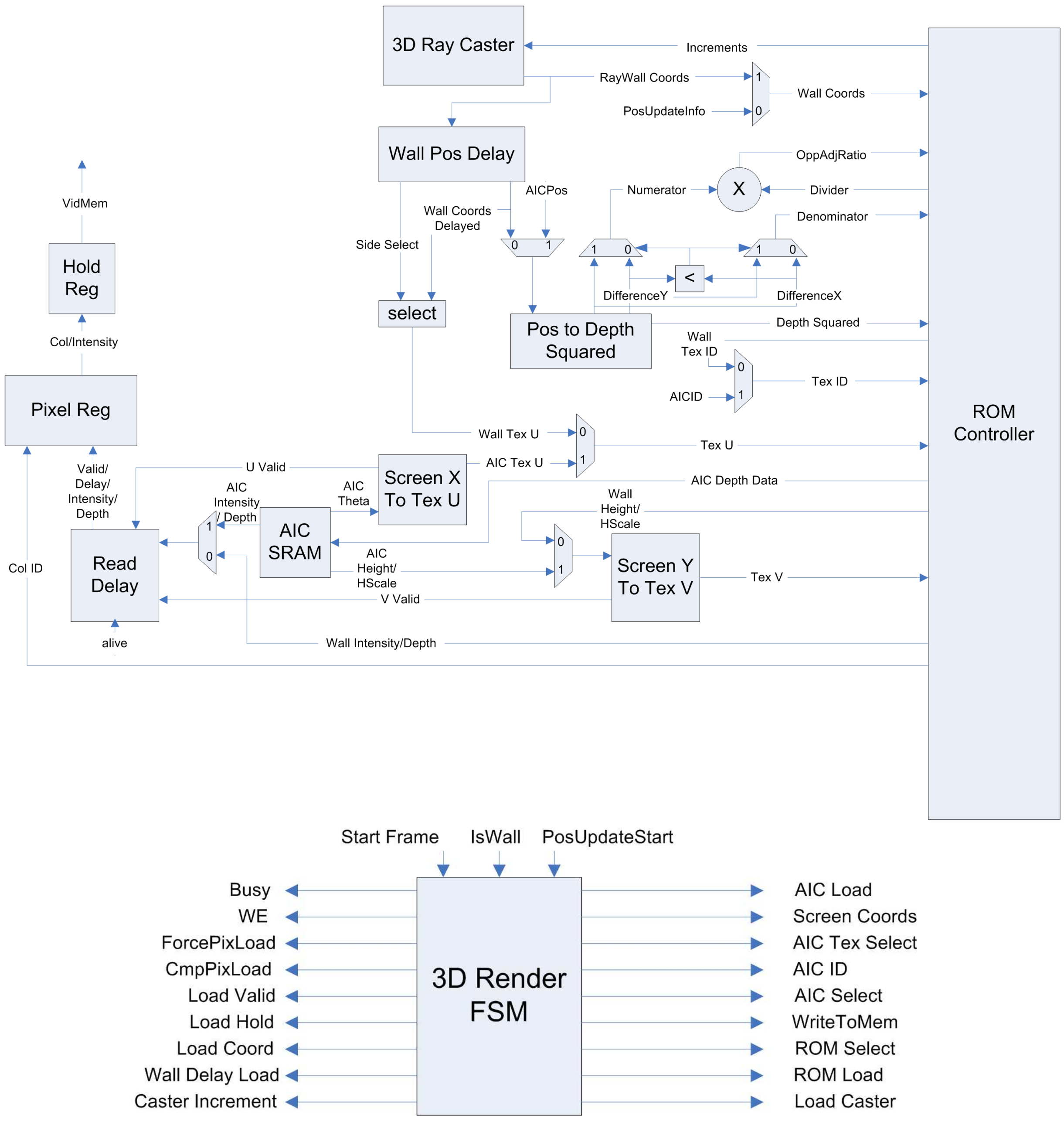

Figure 6: The Block Diagrams for 3D Ray Caster

Table 2: ROM Controller Look-Up Tables

Table Name	Address (Bytes in Hexadecimal)	Size (Bytes in Hexadecimal)	Look-Up Description
Increment 1	00000	00800	Current trace angle to bits 23 to 16 of the increment value
Increment 2	00800	00800	Current trace angle to bits 15 to 8 of the increment value
Increment 3	01000	00800	Current trace angle to bits 7 to 0 of the increment value
Horizontal Width	02000	01000	Sqaure of distance to the angle of an AIC
Square Root 1	03000	01000	Sqaure of distance to bits 11 to 4 of the distance
Height on Screen	04000	01000	Sqaure of distance to the height of the object on screen
Height Scale	05000	01000	Sqaure of distance to the vertical scaling value
Intensity	06000	01000	Sqaure of distance to the intensity scale factor
Divide 1	07000	01000	Divisor to equivalent multiplier bits 11 to 4
Divide 2	08000	01000	Divisor to equivalent multiplier bits 3 to 0
Arctan 1	09000	01000	Opposite over adjacent ratio to angle (via arctangent) bits 11 to 4
Arctan 2	0A000	01000	Opposite over adjacent ratio to angle (via arctangent) bits 3 to 0
Square Root 2	0B000	01000	Sqaure of distance to bits 3 to 0 of the distance
Textures	10000	10000	Texture coordinates to colour for textures 1 to 16
Maps	20000	10000	Map coordinates to texture IDs for maps 1 to 16

2.4.2 3D Ray Caster

The 3D Ray Caster uses the increment values provided by the ROM Controller to calculate the position of wall boundaries. This is done by first calculating the initial boundary positions and then by incrementing these accross the map, choosing the nearest one each time.

2.4.3 Wall Position Delay

The Wall Position Delay simply causes the wall position provided by the 3D Ray Caster to be delayed by two clock cycles.

2.4.4 Position To Depth Squared

Position to Depth Squares converts two sets of coordinates to encoded distance values by calculating the difference between them, squaring these values, adding them, and finally encoding them from 25 bits down to 12 bits. The encoding is carried out by modifying the idea of floating point notation. Bits 2 to 0 contain the exponent (to be explained) and bits 11 to 3 contain the mantissa. The number that is being encoded now has values as follows:

$$D^2 = matissa \times 2^{exponent \times 2}$$

The block diagram can be seen in Fidure 7.

2.4.5 Hold Register

The Hold Register is simply a 16 bit register with a single load.

2.4.6 Pixel Register

The pixel register holds the depth, intensity, and colour ID of the current pixel. Two types of load can be performed: Forced load and Comparitive load. Forced load will always overwirte the contents of the register, either with the current inputs or, if they are invalid, with the internal default values. Comparitive load will only overwrite the current contents if the input is valid and the depth is less than that currently stored. The pixel register only has intensity and colour ID as outputs.

2.4.7 Read Delay

Read Delay forms a delay for data to pass though to simulate the delay of the ROM Accesses.

2.4.8 AIC SRAM

AIC SRAM stores the intensity, depth, angle from the player, height on the screen, and screen height scale for each of the seven AICs (this is an SRAM block with 3 address bits).

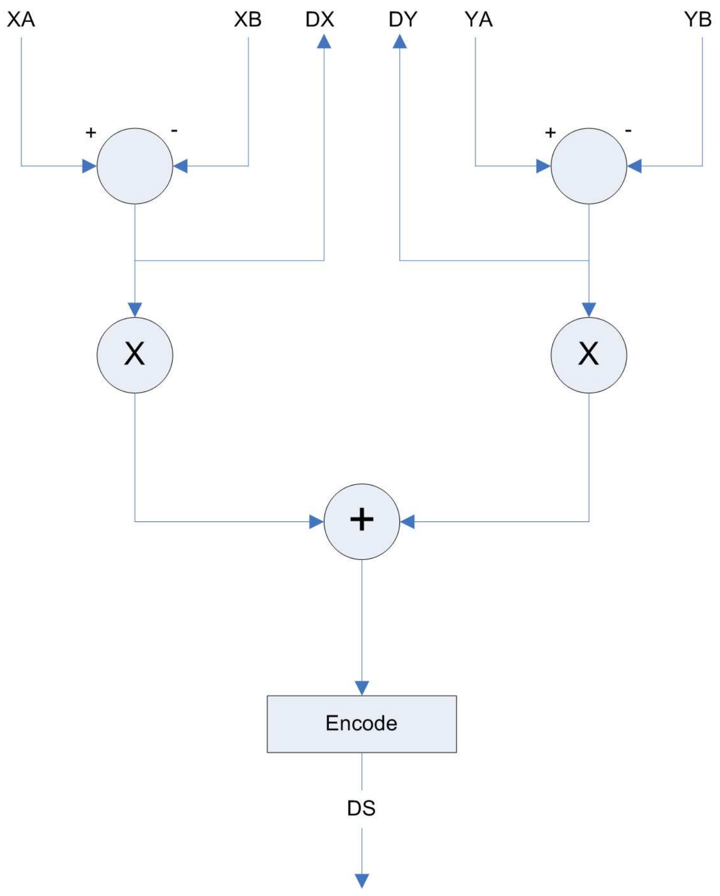

Figure 7: The Position to Depth Squared Block Diagram

2.4.9 Screen X To Texture U

The Screen X To Texture U module converts screen X coordinates into texture U coordinates for the AICs. The conversion is achieved by calculating the difference between the current ray's angle and the angle of the AIC from the user. This difference is then multiplied by the depth and compared with the AIC's width. If it fits the range, it will be converted to a texture coordinate by adding or subtracting a suitable value.

2.4.10 Screen Y To Texture V

The Screen Y To Texture U module converts screen Y coordinates into texture V coordinates for all objects. To achieve the conversion, the current Y coordinate is compared to the object's stored height value to detect whether the coordinates are valid. The actual V coordinate is calculated by multiplying the distance (in pixels) of the current pixel from the centre of the screen by the stored height scale value for the object. The result of the multiplication is converted from an offset from the middle of the texture to an absolute value by adding or subtracting.

2.4.11 3D Render FSM

The 3D Renderer FSM controls the overall operation of the 3D Renderer. It has three basic states, idle, AIC calculations, and drawing. These are not actually the true states, but descriptive groupings (there are actually seven states: idle, three for AIC and three for drawing). The AIC calculations are controlled by a sub-FSM which cycles through each AIC to find its current depth, height, andle, intensity etc. from its current position (as described in the background). The drawing sub-FSM repeats once for each of the 320 columns of pixels. The drawing sub-FSM calculates the distance to the nearest wall along the current direction and then runs the pixel render sub-(sub)-FSM. The pixel render sub-FSM calculates has 29 states and will, for each of the 240 pixels, control calculatiuon of the colour ID and intensity. The top level state transition diagram can be seen in Figure 8.

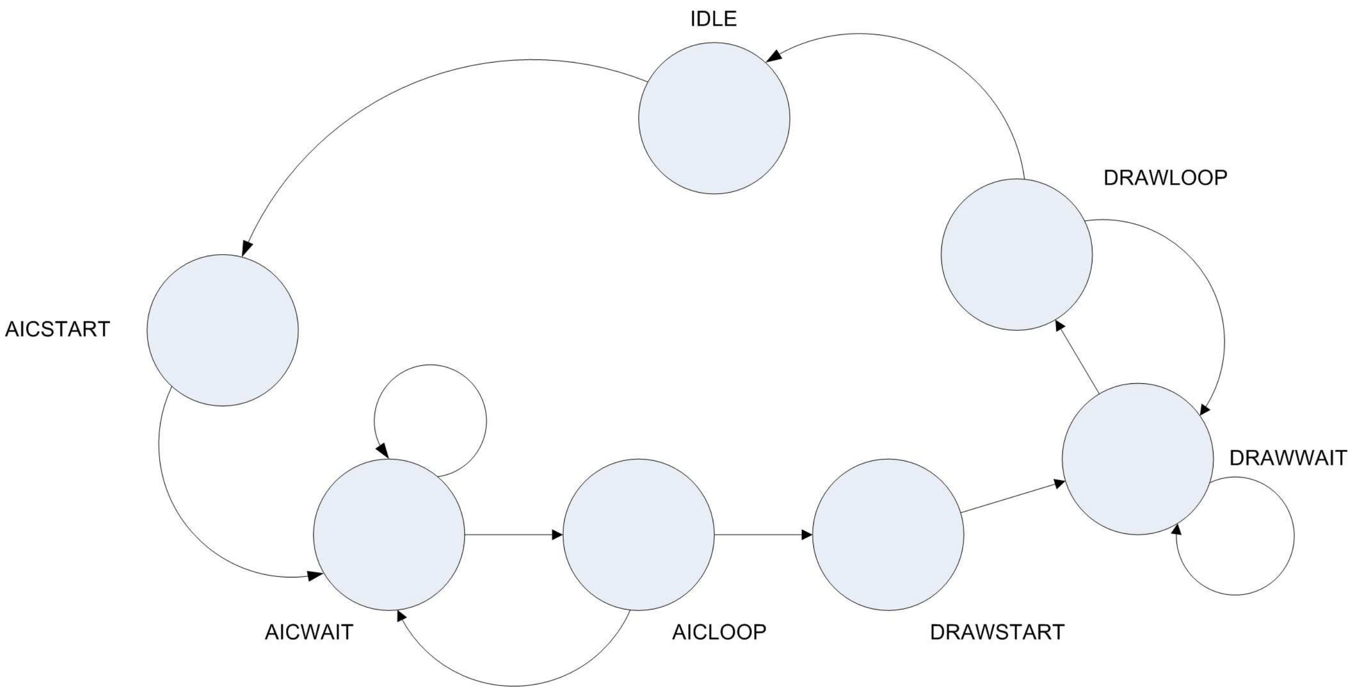

Figure 8: The 3D Renderer FSM State Transition Diagram

2.4.12 3D Renderer Summary

The 3D Renderer is a collection of several simple and complex sub-modules. One of the most complex areas is the pixel render FSM as this controls the timing critical path of the whole renderer. For each pixel, the pixel render FSM will look up the values of eight different textures to get the colours. This means that it needs to achieve $60 \times 320 \times 240 \times 8 = 36864000$ texture lookups every second. The result is that even a single wasted clock cycle or extra texture lookup per pixel is not possible, hence the limit on the numbe of AICs is seven (one texure is used for the walls).

2.5 Player Position (TM)

The player position block simply stores the most recent data about the players position in the maze in terms of a 12-bit x value, 12-bit y value and an 11-bit theta value indicating the orientation. The implementation is nothing more than a 35-bit register therefore, loadable from Position Update and whose outputs link to the game logic, AI and 3D Renderer blocks.

2.6 AIC Positions (TM)

The AIC positions block similarly stores the most recent data about the game characters positions in the maze in terms of a 12-bit x value and a 12-bit y value. A 24 x 8-bit piece of RAM is therefore sufficient for this, which must be addressed by 3-bit character ID supplied by the relevant user block. The outputs are used by the AI, 3D Renderer and Position Update blocks.

2.7 PS/2 Keyboard Interface (TM)

The PS/2 Keyboard interface is comprised of two parts and its function is to receive data sent from the keyboard (upon a user request) and then convert them to a usable internal format for use by the Position Update and Attack modules.

Looking at the specification of a PS/2 keyboard it was decided that communication to the keyboard was unnecessary since the default typematic and repetition rates were suitable for use. This reduced the complexity greatly since only the receive protocol was necessary. A standard device to host transaction involves serially transmitted data synchronised via a 10-16.7 kHz clock line driven by the device. Upon pressing a key the data sent is always 11 bits:

- 1 start bit (always 0)
- 8 data bits, least significant bit first
- 1 parity bit (odd parity)
- 1 stop bit (always 1)

Each bit is sampled by the clock on the falling edge and is changed on the rising edge. Figure 9 shows the clock and lines for the transfer of a single key press.

In order to receive a single byte of data a simple FSM was used to control a shift register to shift in the individual bytes clocked by the falling edge of the ps2Clk. The state transition diagram is shown in Figure 10.

While waiting for a start bit the controller sits in idle, else it begins a bit count and at each falling clock edge will latch in the next data bit received. Upon reaching a count of 9 (therefore the eight data bits and parity bit have been shifted through) the controller tests for the stop bit. If this is true the control code is shifted out of the shift register and compared against a known set of scan codes for standard PS/2 keyboards. The code is saved and a position valid bit is set if a valid directional key is pressed, or attack valid bit is set if it matches the desired attack key, otherwise it is discarded and ignored.

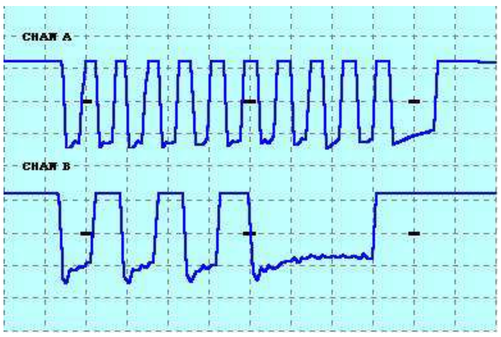

Figure 9: Clock and Data lines for a single PS/2 transfer

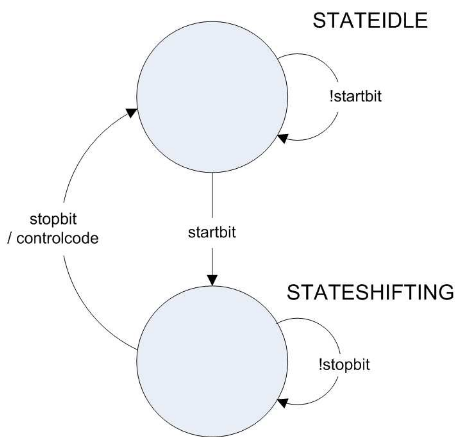

Figure 10: A State Transition Diagram for the PS/2 Controller

The second part of the design is concerned with placing a filter on sampling the clk input any glitches on here could result in invalid data begin latched in, therefore a simple filter requiring a valid logic level change for eight system clock pulses determines a changing clock edge and the correct data to be sampled. The overall block diagram is shown in Figure 11.

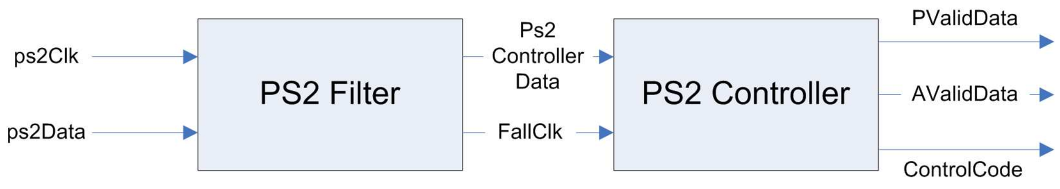

Figure 11: A Block Diagram for the PS/2 Interface

subsectionPosition Update (TM)

The role of position update is to calculate the new positions of the AIC characters and player based on the requests from the AI block and/or the keyboard. It also alters these movements based on the presence of a wall and coordinates attacks via the attack control module. The block diagram is shown in Figure 12.

The Position Calc FSM is where the bulk of the work is done, cycling through each player and AI and

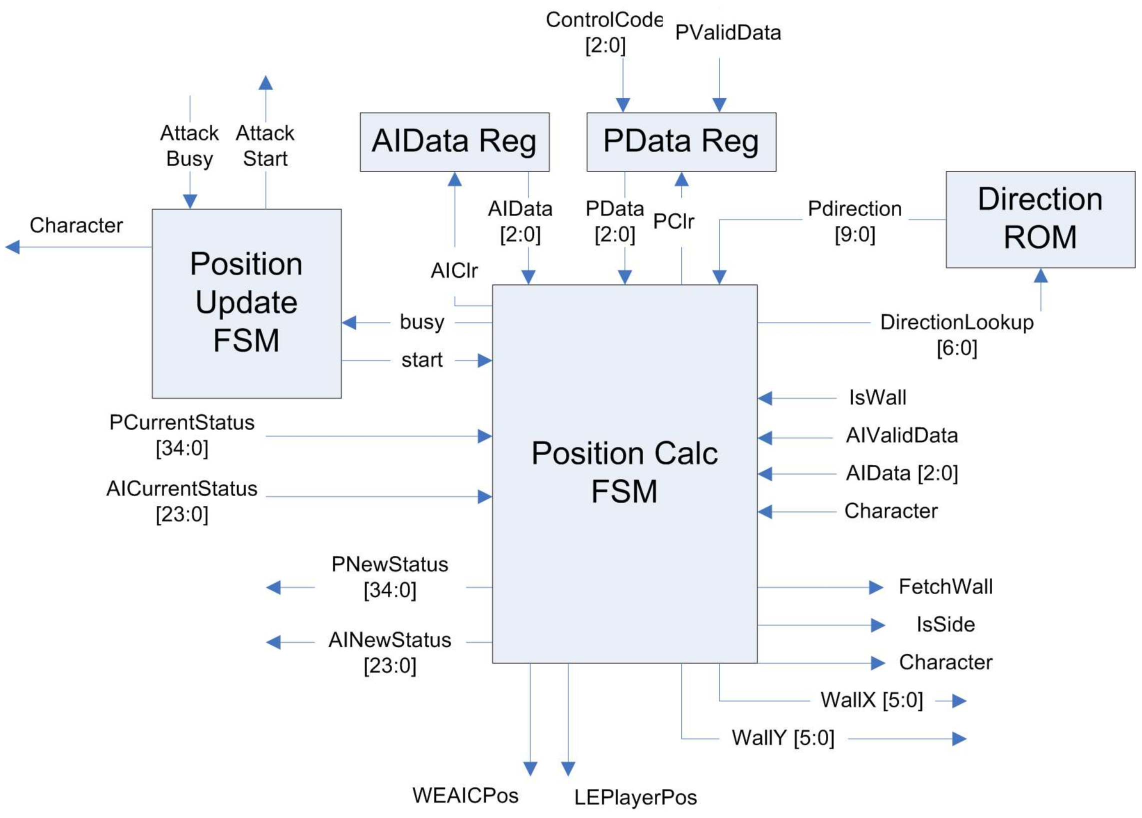

Figure 12: A Block Diagram for the Position Update module

forming the new positions, checking for walls before outputting these to the relevant registers. From idle it moves into a current position state where the Player Position or AIC Position memories are read for the current information. In the next state the request from the keyboard (AI block for AICs) is converted into a directional request and a from this a change in position is calculated. This differs between player and AICs. The player can have an orientation, so the top seven bits of the angle are used to form a change in x and y positions based on a lookup for a standard forward movement for that angle. The AICs can only move in one of eight directions and the one to minimise the distance between itself and the player (information provided by the ray caster) is used to form the required movement. The x coordinate is then added and a check for the nearest side wall is made, followed by a similar check for the y coordinate and top walls, thus allowing a new position to be specified for the relevant character. In the presence of walls there can be no movement in that direction. The new positions are then written to the memory stores and the process can begin again.

The AI Data and P Data Registers simply hold the inputs from the respective blocks for use by Position Calc FSM (i.e. the keyboard or AI block) . Each supplies a 3-bit movement code, specifying the eight directions for the AI character or the four directions plus two turns for the player. The ROM stores the standard changes in x and y positions for a given orientation as viewed in the forward direction. Finally the Position Update FSM simply switches between the position updates and the coordination of attack control for player and AICs to ensure all requests are met each time around.

2.8 Health Memory (TM)

The health memory is a piece of RAM that stores the health points for the player and all AICs. In the event of a successful attack the health points are read out of memory by attack control and decremented before being written back into memory. The 3D Renderer consistently reads the memory for any health reaching zero whereupon it will remove the character from the game.

2.9 Attack Control (TM)

Attack control receives attack commands from the AI or keyboard and then alters the health accordingly. The state transition diagram of operation is shown in Figure 13. From idle it waits for the start signal to be asserted by Position Update before moving into read health. Here, the attack status register is used to determine whether an attack has been asserted (effectively an 8 x 1-bit memory block) by polling the relevant bit addressed by the character ID. If this is 0 and there is no attack it will return to idle else it will read the relevant health level (addressed by the same character ID) and decrement it before writing it back into memory in the same position and returning to idle.

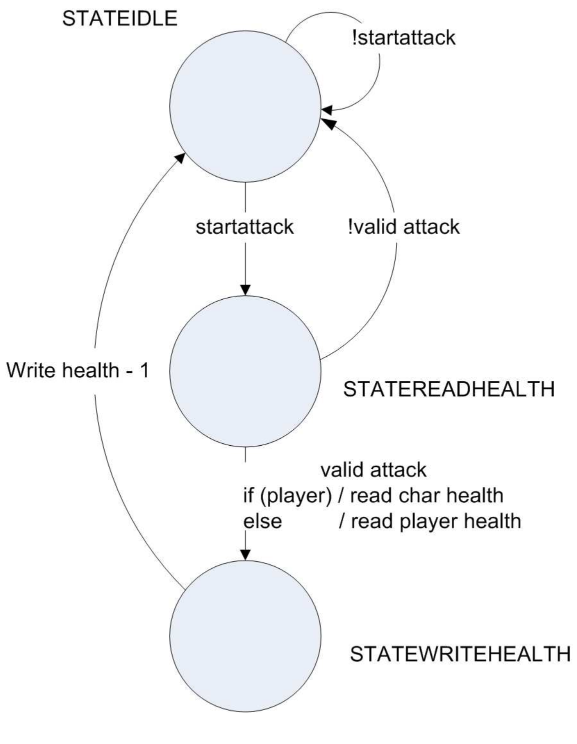

Figure 13: A State Transition Diagram for the Attack Control Module

2.10 Game Logic

2.11 A.I. (TM)

The AI module controls the movements of the game characters and causes their attacks if close enough to the player. The state transition diagram of the AI Block is shown in Figure 14. From idle upon receiving a request to calculate new AI positions the AI block reads the current AI depth for the given character (i.e. the distance of the character from the player) and the angle from players viewpoint. From the angle it then assigns the new direction in which the AIC aims to move (irrespective of walls), one of eight directions as shown:

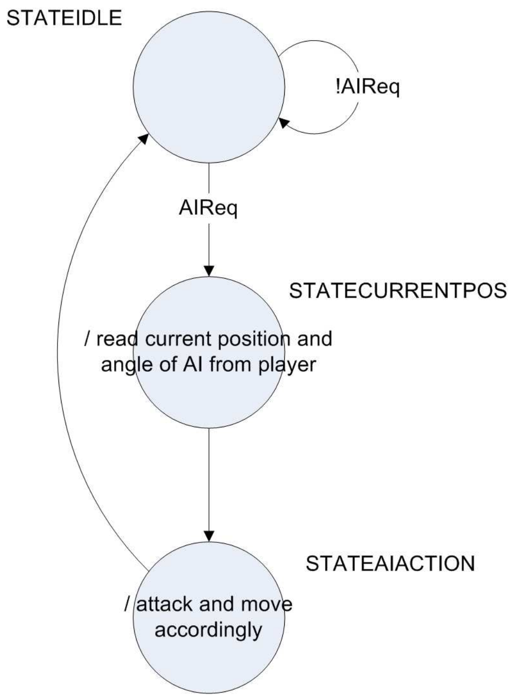
STATEIDLE
!AIReq
AIReq
STATECURRENTPOS
/ read current position and angle of AI from player
STATEAIACTION
/ attack and move accordingly

Figure 14: A State Transition Diagram for the AI Module

Reading AIC Depth (from the 3D Renderer) then determines how close the character is to the player and governs the attack if below a certain value. The relevant values are returned to Position Update in the calculation of new positions.

3 Testing and Debugging

Testing and debugging go hand in hand: tests are carried out to check for behavioral errors and debugging removes these. Testing simply involves trying out the test target in a variety of circumstances designed to present any possible errors in the system. Once such a bug has been found (which, when putting a complex system together happens all too regularly), debugging techniques are employed to remove these errors (or *bugs*). Unfortunaley debugging is never easy for hardware and it only gets harder as the complexity of the system increases.

In this section the general strategy, some methods used and some actual bugs found will be described.

3.1 The General Strategy

Testing and debugging are carried out in two main stages, namely *Simulation* and *Hardware Testing.*

In simulation, the computer is used to simulate the behavior of individual modules and groups of modules. These simulations allow the removal of many of the bugs in simple to medium complexity modules. Unfortunately, simulations are not very useful for certain modules, especially those with vary large amounts of data or those with very long running times.

After all the suitable modules have been simulated and their bugs removed, simple collections of modules can be tested in hardware. It is important to test them in small groups so that bugs can quickly be found. Testing should always begin with the output modules.

3.2 Debugging Methods Used

3.2.1 Partial Implementations and Firm Foundations

When a bug is found in a complex test, first make sure that those modules that are thought to be working correctly actually are. The second step is to remove modules and disable features until the module behaves as expected, and then to add them back one by one until it no longer works. Thus the last feature that is added is the one that is either causing the problem or displaying the problem.

3.2.2 Error Manipulation, Isolation and Test Programs

Once the rough location of a bug is known (to within a few modules), some parameter or signal must be found which causes the problem to vary, it can then be traced to the problem. Often in order to track these down, several modules must be distinguished between. A very useful technique to achieve this is to write a simple test module that emulates the original in all ways important to the problem (usually accepting inputs, but not doing manipulation in order to test control signals).

3.3 Debugging Termanator 3D

The functionality of the design was built up slowly such that complexity could be added stage by stage. Whilst all blocks could be simulated simply, the real system was built in many stages to ensure working interfaces for the VGA and keyboard before testing the 3D engine and the game logic beyond. The scope of debugging was reduced using this method and proved invaluable in narrowing down error searches. Simple blocks were also recycled from previous laboratories that ensured their stability in this environment.

3.3.1 Simulation

The implementation in Verilog was first tested in ModelSim and the beauty of modularisation internally was that each block could be tested individually by asserting any inputs manually and analysing the behaviour. The simple sub modules were easily simulated with reset and input levels changed by hand in a waveform generator. The FSMs were simple to simulate and all control signals were successfully imitated for a working system.

3.3.2 VGA Interface Testing

The first part of the test schedule was to design and build a working VGA Interface to ensure a stable visible output. Timing was crucial for this process, to ensure that given a pattern of coloured vertical bars could be depicted on screen. This worked efficiently quite quickly and provided confidence to tackle the integration of the 3D Renderer.

3.3.3 PS/2 Interface Testing

In a separate part of the test schedule the PS/2 Interface was easily tested by reading in a relevant scan code and displaying it on the seven segment display of the Spartan 3 Starter Board. The test was set up so that only relevant codes were displayed and a delay was introduced so that they were held for a visible length of time.

3.3.4 Video Memory Access Controller

This was tested by providing a test module to emulate the 3D Renderer by producing coloured bar outputs. This module contained the most subtle bug found, where past written values were read from a high impedance data wire.

3.3.5 3D Renderer

This was tested by first using the FSM for timing of memory writes of coloured bars, then by adding in various sections until a system that worked sufficiently was produced so that other modules could be tested.

4 Conclusions

The project aims were met with some success, in that a working 3D maze was created and could be navigated by a user. The debugging was very successful, but not enough time was allowed for the process, so that many bugs still remain and much of the code has not been integrated. Had the VGA Controller been scheduled to be implemented earlier, the other parts of the project might have been debugged in a more parallel fashion. Additionally the DAC was received fairly late and as such the VGA could not be tested in hardware much earlier. However the final outcome was a pleasing success with working user interfaces and a 3D image on screen minus the AI characters.

References

[1] Chapweske, A, PS/2 Mouse/Keyboard Protocol, (1999) which may be found at: http://panda.cs.ndsu.nodak.edu/ achapwes/PICmicro/PS2/ps2.htm

[2] Analog Devices, ADV7120 DAC Datasheet

APPENDIX

A Verilog Code

```
/**************************************************************************************************
*                                                                                                 *
*  Filename:        m_AI.v                                                                        *
*  Description:     Controls the new AI desired positions and attack                              *
*  Date:            05/12/04                                                                      *
*  Author:          Tim Mamtora                                                                   *
*  Notes:           05/12/04 17.24 First Draft                                                    *
*                                                                                                 *
*                                                                                                 *
**************************************************************************************************/


module m_AI(clk, reset, i_AIReq, i_Character, i_AICDepth, i_AICTheta, o_Character, o_AIDirection,
                         o_AIAttack, o_AIBusy, r_State, r_AICDepth, r_AICTheta);

   // state parameters
   parameter                 p_STATE_IDLE            = 0;
   parameter                 p_STATE_CURRENTPOS      = 1;
   parameter                 p_STATE_AIACTION        = 2;

   // module parameters
   parameter                 p_LEFT                  = 1;
   parameter                 p_RIGHT                 = 2;
   parameter                 p_UP                    = 3;
   parameter                 p_DOWN                  = 4;
   parameter                 p_UPLEFT                = 5;
   parameter                 p_UPRIGHT               = 6;
   parameter                 p_DOWNLEFT              = 7;
   parameter                 p_DOWNRIGHT             = 8;

   // module inputs
   input                     clk;
   input                     reset;
   input                     i_AIReq;
   input [2:0]               i_Character;
   input [11:0]              i_AICDepth;    // straight distance from player to AI character
   input [10:0]              i_AICTheta;    // angle from player viewpoint to AI character from 3D Renderer

   // module outputs
   output [2:0]              o_AIDirection; // change request for AI character position
   output                    o_AIAttack;    // attack request for AI character
   output                    o_AIBusy;
   output [2:0]              o_Character;

   // debug
   output [1:0]              r_State;
   output [11:0]             r_AICDepth;
   output [10:0]             r_AICTheta;

   // internal registers
   reg [1:0]                 r_State;
   reg [11:0]                r_AICDepth;
   reg [10:0]                r_AICTheta;


   // output regsiters
   reg [2:0]                 o_AIDirection;
   reg                       o_AIAttack;
   reg                       o_AIBusy;
   reg [2:0]                 o_Character;

   always @ (posedge clk or posedge reset)
   begin
      if (reset)
      begin
         o_Character    <= 0;
         o_AIDirection  <= 0;
         o_AIBusy       <= 0;
         r_State        <= p_STATE_IDLE;
      end // if statement
      else
      begin
         case (r_State)

         p_STATE_IDLE:
         begin
            o_Character    <= 0;
            o_AIDirection  <= 0;
            o_AIBusy       <= 0;
            if (i_AIReq == 1)
               r_State  <= p_STATE_CURRENTPOS;
            else
               r_State  <= p_STATE_IDLE;
         end // p_STATEIDLE case

         p_STATE_CURRENTPOS:  // read current depth and angle of character from player
         begin
            o_AIBusy                <= 1;
            o_Character             <= i_Character;
            r_AICDepth              <= i_AICDepth;
            r_AICTheta              <= i_AICTheta [10:4];
            r_State                 <= p_STATE_AIACTION;
         end // p_STATECURRENTPOS case

         p_STATE_AIACTION:    // assign direction of movement and attack
         begin
            o_AIBusy                <= 1;
            if ((r_AICDepth[11:6] | 6'b0) == 0)
               o_AIAttack  <= 1;
            if (r_AICTheta >= 7'd116)
               o_AIDirection  <= p_RIGHT;
            else if (r_AICTheta > 7'd106)
               o_AIDirection  <= p_UPRIGHT;
            else if (r_AICTheta >= 7'd84)
               o_AIDirection  <= p_UP;
            else if (r_AICTheta > 7'd74)
               o_AIDirection  <= p_UPLEFT;
            else if (r_AICTheta >= 7'd52)
               o_AIDirection  <= p_LEFT;
            else if (r_AICTheta > 7'd42)
               o_AIDirection  <= p_DOWNLEFT;
            else if (r_AICTheta >= 7'd20)
               o_AIDirection  <= p_DOWN;
            else if (r_AICTheta > 7'd10)
               o_AIDirection  <= p_DOWNRIGHT;
            else
               o_AIDirection  <= p_RIGHT;
            r_State  <= p_STATE_IDLE;
         end  // p_STATEAIACTION case
         default: r_State <= p_STATE_IDLE;
         endcase
      end // else statement
   end // always block
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:         m_AttackControl.v                                                              *
*  Description:      Controls the AI character and player attacks                                   *
*  Date:             03/12/04                                                                       *
*  Author:           Tim Mamtora                                                                    *
*  Notes:            03/12/04 17:02 First Draft                                                     *
*                                                                                                   *
****************************************************************************************************/

module m_AttackControl(clk, reset, i_AIAttack, i_ADataReady, i_AttackCharacterIn, i_StatusClear,
                       i_AttackStart, i_CCID, i_health, o_AttackBusy, o_health, o_AttackCharacterOut, o_we);
   // parameters - none

   // module inputs
   input          clk;
   input          reset;
   input          i_AIAttack;
   input          i_ADataReady;
   input [2:0]    i_AttackCharacterIn;
   input          i_StatusClear;
   input          i_AttackStart;
   input [2:0]    i_CCID;
   input [3:0]    i_health;

   // module outputs
   output         o_AttackBusy;
   output [3:0]   o_health;
   output [2:0]   o_AttackCharacterOut;
   output         o_we;

   // internal wires
   wire [7:0]     w_AttackStatus;

   // instantiations
   m_AttackReg          s_AttackReg(clk, i_AIAttack, i_ADataReady, i_AttackCharacterIn,
                                      i_StatusClear, w_AttackStatus);
   m_AttackControlFsm   s_AttackControlFsm(clk, reset, i_AttackStart, w_AttackStatus,
                                      i_AttackCharacterIn, i_CCID, i_health, o_AttackBusy,
                                      o_health, o_AttackCharacterOut, o_we);

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_AttackControlFsm.v                                                            *
*  Description:     Controls the attacks from the player and AI characters                          *
*  Date:            30/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           30/11/04 21:54 First Draft                                                      *
*                   03/12/04 10.55 Modified FSM                                                     *
*                                                                                                   *
****************************************************************************************************/


module m_AttackControlFsm(clk, reset, i_AttackStart, i_AttackStatus, i_AttackCharacterIn, i_CCID, i_health,
                          o_AttackBusy, o_health, o_AttackCharacterOut, o_we);

   // state parameters
   parameter                  p_STATE_IDLE        = 0;
   parameter                  p_STATE_READHEALTH  = 1;
   parameter                  p_STATE_WRITEHEALTH = 2;

   // module parameters
   parameter                  p_HEALTHWIDTH       = 4;
   parameter                  p_PLAYERID          = 0;

   // module inputs
   input                      clk;
   input                      reset;
   input                      i_AttackStart;
   input [2:0]                i_AttackCharacterIn;  // requested AI character attack
   input [2:0]                i_CCID;               // centre character ID, i.e. a character or a
                                                    //  wall(miss) for a player attack
   input [(p_HEALTHWIDTH-1):0]   i_health;          // current health
   input [7:0]                i_AttackStatus;       // status register storing the attack requests for
                                                    //  player and AICs

   // module outputs
   output                     o_AttackBusy;
   output [(p_HEALTHWIDTH-1):0]  o_health;
   output [2:0]               o_AttackCharacterOut;
   output                     o_we;                 // write enable for health register
```

```
// internal registers
reg [1:0]                    r_State;
reg [(p_HEALTHWIDTH-1):0]    r_health;
reg [7:0]                    r_AttackStatus;
reg [2:0]                    r_CCID;

// output registers
reg                          o_AttackBusy;
reg [(p_HEALTHWIDTH-1):0]    o_health;
reg [2:0]                    o_AttackCharacterOut;
reg                          o_we;

always @ (posedge clk or posedge reset)
begin
   if (reset)
   begin
      r_State              <= p_STATE_IDLE;
      o_AttackBusy         <= 0;
      o_health             <= 0;
      o_AttackCharacterOut <= 0;
      o_we                 <= 0;
      r_AttackStatus       <= 0;
   end // if statement
   else
   begin
      case (r_State)

      p_STATE_IDLE:
      begin
         o_AttackBusy         <= 0;
         o_health             <= 0;
         o_AttackCharacterOut <= 0;
         o_we                 <= 0;
         r_AttackStatus       <= 0;
         if (i_AttackStart == 1)
         begin
            r_State              <= p_STATE_READHEALTH;
            r_AttackStatus       <= i_AttackStatus;
            r_CCID               <= i_CCID;
         end // if statement
         else
         begin
            r_State              <= p_STATE_IDLE;
         end // else statement
      end // p_STATEIDLE case

      p_STATE_READHEALTH:
      begin
         o_AttackBusy   <= 1;
         if (r_AttackStatus [i_AttackCharacterIn] == 0)
         begin
            r_State  <= p_STATE_IDLE;
         end // if statement
         else
         begin
            if (i_AttackCharacterIn == 0) // Player Attack
            begin
               if (r_CCID != 0)
               begin
                  o_AttackCharacterOut <= r_CCID;
                  o_we                 <= 0;
                  r_health             <= i_health;   // read in current health of character
                  r_State              <= p_STATE_WRITEHEALTH;
               end // if statement
               else
               begin
                  o_AttackCharacterOut <= 0;
                  o_we                 <= 0;
                  r_health             <= 0;          // do nothing, the attack is a miss
                  r_State              <= p_STATE_IDLE;
               end
            end // if statement
```

```
               else                        // Character Attack
               begin
                  o_AttackCharacterOut <= 0;
                  o_we                 <= 0;
                  r_health             <= i_health;   // read in current health of player
                  r_State              <= p_STATE_WRITEHEALTH;
               end // else statement
            end // else statement
         end // p_STATEREADHEALTH case

         p_STATE_WRITEHEALTH:
         begin
            o_AttackBusy         <= 1;
            o_health             <= r_health - 1;  // decrement health
            o_AttackCharacterOut <= o_AttackCharacterOut;
            o_we                 <= 1;             // write to health memory
            r_State     <= p_STATE_IDLE;
         end // p_STATENEWHEALTH case

         default: r_State  <= p_STATE_IDLE;
         endcase
      end // else statement
   end // always block
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:         m_AttackReg.v                                                                  *
*  Description:      Attack Register                                                                *
*  Date:             03/12/04                                                                       *
*  Author:           Tim Mamtora                                                                    *
*  Notes:            03/12/04 11:51 First Draft                                                     *
*                                                                                                   *
****************************************************************************************************/

module m_AttackReg(clk, i_AIAttack, i_ADataReady, i_AttackCharacterIn, i_StatusClear, o_AttackRegOut);

   // parameters - none

   // module inputs
   input             clk;
   input             i_AIAttack;
   input             i_ADataReady;          // Player Attack
   input [2:0]       i_AttackCharacterIn;   // Character Attack
   input             i_StatusClear;

   // module outputs
   output [7:0]      o_AttackRegOut;

   // internal registers - none

   // output registers
   reg [7:0]         o_AttackRegOut;

   always @ (posedge clk)
   begin
      if (i_StatusClear)
         o_AttackRegOut <= 0;
      if (i_ADataReady == 1)
         o_AttackRegOut[0] <= 1;
      if (i_AIAttack == 1)
         o_AttackRegOut[i_AttackCharacterIn] <= 1;
   end
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:         m_PDataReg.v                                                                   *
*  Description:      8-bit Register                                                                 *
*  Date:             03/12/04                                                                       *
*  Author:           Tim Mamtora                                                                    *
*  Notes:            03/12/04 20:28 First Draft                                                     *
*                                                                                                   *
*****************************************************************************************************/

module m_PDataReg(clk, i_ld, i_clr, i_d, o_q);

   // parameters - none

   // module inputs
   input            clk;
   input            i_ld;
   input            i_clr;
   input [7:0]      i_d;

   // module outputs
   output [7:0]     o_q;

   // internal registers - none

   // output registers
   reg [7:0]        o_q;

   always @ (posedge clk)
   begin
      if(i_clr)
         o_q <= 0;
      else if(i_ld)
         o_q <= i_d;
   end
endmodule

/*****************************************************************************************************
*                                                                                                   *
*  Filename:        m_PixelFsm.v                                                                    *
*  Description:     Controls the pixel data acquisition from the video memory                       *
*  Date:            28/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           28/11/04 19:19 First Draft                                                      *
*                                                                                                   *
*****************************************************************************************************/



module m_PixelFsm(clk, reset, i_MemBusy, o_PixelBusy, o_MemStart, o_LEColIDReg, o_LEIntensityValReg,
                                o_PixelError, o_vgaout);

   // state parameters
   parameter           p_STATE_IDLE      = 0;
   parameter           p_STATE_READREQ   = 1;
   parameter           p_STATE_READWAIT  = 2;
   parameter           p_STATE_READWAIT2 = 3;
   parameter           p_STATE_READRESET = 4;


   // module inputs
   input               clk;
   input               reset;
   input               i_MemBusy;         // Memory Controller Busy

   // module outputs
   output              o_PixelBusy;
   output              o_MemStart;        // Request read from Memory Controller
   output              o_LEColIDReg;
   output              o_LEIntensityValReg;
   output              o_PixelError;
   output              o_vgaout;

   // output registers
   reg                 o_PixelBusy;
   reg                 o_MemStart;
   reg                 o_LEColIDReg;
   reg                 o_LEIntensityValReg;
   reg                 o_PixelError;
   reg                 o_vgaout;

   // internal registers
   reg [2:0]           r_State;

      always @ (posedge clk or posedge reset)
      begin
         if (reset)
         begin
            // defaults
            r_State              <= p_STATE_IDLE;
            o_PixelBusy          <= 0;
            o_MemStart           <= 0;
            o_LEColIDReg         <= 0;
            o_LEIntensityValReg  <= 0;
            o_PixelError         <= 0;
            o_vgaout             <= 0;
         end // if statement
         else
         begin
            case (r_State)

            p_STATE_IDLE:
            begin
               o_PixelBusy          <= 0;
               o_MemStart           <= 0;
               o_LEColIDReg         <= 0;
               o_LEIntensityValReg  <= 0;
               o_PixelError         <= 0;
               o_vgaout             <= 0;
               r_State              <= p_STATE_READREQ;
            end   // p_STATEIDLE case

            p_STATE_READREQ:
            begin
               o_PixelBusy          <= 1;
               o_MemStart           <= 1;
               o_LEColIDReg         <= 0;
               o_LEIntensityValReg  <= 0;
               o_PixelError         <= 0;
               o_vgaout             <= 0;
               r_State              <= p_STATE_READWAIT;
            end   // p_STATEREADREQ case

            p_STATE_READWAIT:       // Wait for Memory Controller to stop being busy
            begin
               o_PixelBusy          <= 1;
               o_MemStart           <= 0;
               o_PixelError         <= 0;
               o_vgaout             <= 0;
               if (i_MemBusy == 1)
               begin
                  o_LEColIDReg         <= 0;       // continue waiting...
                  o_LEIntensityValReg  <= 0;
                  r_State              <= p_STATE_READRESET;
               end // if statement
               else
               begin
                  o_LEColIDReg         <= 1;       // load in read results
                  o_LEIntensityValReg  <= 1;
                  r_State              <= p_STATE_READWAIT2;
               end // else statement
            end   // p_STATEREADWAIT case

            p_STATE_READWAIT2:
            begin
               o_PixelBusy             <= 1;
               o_MemStart              <= 0;
               o_PixelError            <= 0;
               o_LEColIDReg            <= 0;
               o_LEIntensityValReg     <= 0;
               o_vgaout                <= 1;
               r_State                 <= p_STATE_IDLE;
            end   // p_STATEREADWAIT2

            p_STATE_READRESET:
            begin
               o_PixelBusy             <= 1;
               o_MemStart              <= 0;
               o_vgaout                <= 1;
               if (i_MemBusy == 1)
               begin
                  o_LEColIDReg         <= 0;
                  o_LEIntensityValReg  <= 0;
                  o_PixelError         <= 1;       // Memory Controller should return read after max 2 clock cycles
                  r_State              <= p_STATE_IDLE;
               end // if statement
               else
               begin
                  o_LEColIDReg         <= 1;       // load in read results
                  o_LEIntensityValReg  <= 1;
                  o_PixelError         <= 0;
                  r_State              <= p_STATE_IDLE;
               end // else statement

            end   // p_STATEREADRESET case

            default:    r_State <= p_STATE_IDLE;
            endcase
         end   // else statement
      end // always atatement
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_PlayerPosition.v                                                              *
*  Description:     35-bit Register for Px, Py and angle T                                          *
*  Date:            30/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           30/11/04 22:01 First Draft                                                      *
*                                                                                                   *
****************************************************************************************************/

module m_PlayerPosition(clk, i_ld, i_d, o_q);

   // parameters - none

   // module inputs
   input            clk;
   input            i_ld;
   input [34:0]     i_d;

   // module outputs
   output [34:0]    o_q;

   // internal registers - none

   // output registers
   reg [34:0]       o_q;

   always @ (posedge clk)
   begin
      if(i_ld)    o_q <= i_d;
   end
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_ps2.v                                                                         *
*  Description:     Controls the interface with the keyboard game controls                          *
*  Date:            15/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           15/11/04 21:19 First Draft                                                      *
*                                                                                                   *
```

```
*****************************************************************************************************/

module m_ps2(clk, reset, i_ps2clk, i_ps2data, o_PValidData, o_AValidData, o_ControlCode);

   // module inputs
   input                clk;
   input                reset;
   input                i_ps2clk;
   input                i_ps2data;

   // module outputs
   output               o_PValidData;       // Valid position data flag
   output               o_AValidData;       // Valid attack data flag
   output [7:0]         o_ControlCode;      // Direction request code

   // debug
   // output [6:0]         o_Segment;
   // output [3:0]         o_AN;

   // internal wires
   wire                 w_ps2ControllerData;
   wire                 w_FallClk;

   // debug
   // wire                 w_PValidData;
   // wire                 w_AValidData;
   // wire [7:0]           w_ControlCode;
   // wire [7:0]           w_DisplayCode;


   m_ps2Filter          s_ps2Filter(clk, reset, i_ps2clk, i_ps2data, w_ps2ControllerData, w_FallClk);
   m_ps2Controller      s_ps2Controller(clk, reset, w_FallClk, w_ps2ControllerData, o_PValidData,
                                             o_AValidData, o_ControlCode);

   // debug
   // m_Delay              s_Delay(clk, reset, w_PValidData, w_AValidData, w_ControlCode, w_DisplayCode,
                                     o_PValidData, o_AValidData);
   // m_SevenSegmenttest   s_SevenSegment(clk, reset, w_DisplayCode, o_Segment, o_AN);

endmodule

/*****************************************************************************************************
*                                                                                                    *
*  Filename:        m_ps2Controller.v                                                                *
*  Description:     Controls the interface with the keyboard game controls                           *
*  Date:            15/11/04                                                                         *
*  Author:          Tim Mamtora                                                                      *
*  Notes:           15/11/04 21:06 First Draft                                                       *
*                   02/12/04 17.45 o_AValidData added                                                *
*                                                                                                    *
*****************************************************************************************************/


module m_ps2Controller(clk, reset, i_FallClk, i_ps2ControllerData, o_PValidData, o_AValidData, o_ControlCode);

   // state parameters
   parameter            p_STATE_IDLE      = 0;
   parameter            p_STATE_SHIFTING  = 1;

   // controller code parameters
   parameter            p_LEFT         = 8'h1A; // Z key
   parameter            p_RIGHT        = 8'h22; // X key
   parameter            p_UP           = 8'h2D; // R key
   parameter            p_DOWN         = 8'h23; // D key
   parameter            p_TURNLEFT     = 8'h24; // E key
   parameter            p_TURNRIGHT    = 8'h2C; // T key
   parameter            p_ATTACK       = 8'h29; // SPACE

   parameter            p_CONTROLLEFT        = 1;
   parameter            p_CONTROLRIGHT       = 2;
   parameter            p_CONTROLUP          = 3;
   parameter            p_CONTROLDOWN        = 4;
   parameter            p_CONTROLTURNLEFT    = 5;
   parameter            p_CONTROLTURNRIGHT   = 6;
   // parameter            p_CONTROLATTACK      = 0;

   // module inputs
   input                clk;
   input                reset;
   input                i_FallClk;
   input                i_ps2ControllerData;

   // module outputs
   output               o_PValidData;
   output               o_AValidData;
   output [7:0]         o_ControlCode;

   // internal registers
   reg [8:0]            r_ShiftReg;
   reg [3:0]            r_BitCount;
   reg                  r_State;
   reg [7:0]            r_ScanCode;


   always @ (posedge clk or posedge reset)
   begin
      if(reset)
      begin
         // default assignments
         r_State        <= p_STATE_IDLE;
         r_BitCount     <= 0;
         r_ShiftReg     <= 0;
         r_ScanCode     <= 0;
      end // if (reset) statement
      else
      begin
         case (r_State)

            p_STATE_IDLE:
            begin
               r_BitCount  <= 0;
               r_ScanCode  <= 0;
               if ((i_FallClk == 1) && (i_ps2ControllerData == 0)) // start bit
                  r_State <= p_STATE_SHIFTING;
               else r_State <= p_STATE_IDLE;
            end //  p_STATEIDLE case

            p_STATE_SHIFTING:
            begin
               if (r_BitCount == 9)
               begin
                  if (i_FallClk == 1)  // stop bit
                  begin
                     r_ScanCode     <= r_ShiftReg[7:0];
                     r_State        <= p_STATE_IDLE;
                  end // if statement
               end // if statement
               else if (i_FallClk == 1)
               begin
                  r_BitCount <= r_BitCount + 1;
                  r_ShiftReg <= {i_ps2ControllerData, r_ShiftReg[8:1]};    // shift in scan code (right shift)
               end // else if statement
            end // p_STATESHIFTING case

            default:    r_State <= p_STATE_IDLE;
         endcase
      end // else statement

   end // always block

   assign o_ControlCode =  (r_ScanCode == p_LEFT)     ? p_LEFT :
                           (r_ScanCode == p_RIGHT)    ? p_RIGHT:
                           (r_ScanCode == p_UP)       ? p_UP:
                           (r_ScanCode == p_DOWN)     ? p_DOWN:
                          (r_ScanCode == p_TURNLEFT) ? p_TURNLEFT:
                          (r_ScanCode == p_TURNRIGHT)? p_TURNRIGHT: 0;
   assign o_PValidData =  ((r_ScanCode == p_LEFT) | (r_ScanCode == p_RIGHT) | (r_ScanCode == p_UP) |
                           (r_ScanCode == p_DOWN) | (r_ScanCode == p_TURNLEFT) | (r_ScanCode == p_TURNRIGHT))? 1: 0;
   assign o_AValidData =  (r_ScanCode == p_ATTACK) ? 1 : 0;

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_ps2Filter.v                                                                   *
*  Description:     Filters the PS/2 clock line to ensure robust transfers                          *
*  Date:            15/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           15/11/04 20:21 First Draft                                                      *
*                                                                                                   *
****************************************************************************************************/

module m_ps2Filter(clk, reset, i_ps2clk, i_ps2data, o_ps2ControllerData, o_FallClk);

   // parameters
   parameter                  p_FILTERSIZE   = 8;

   // module inputs
   input                      clk;
   input                      reset;
   input                      i_ps2clk;
   input                      i_ps2data;

   // module outputs
   output                     o_ps2ControllerData;
   output                     o_FallClk;

   // internal registers
   reg [(p_FILTERSIZE-1):0]   r_FilterReg;
   reg                        r_FilterClk;

   // output registers
   reg                        o_FallClk;
   reg                        o_ps2ControllerData;

   always @ (posedge clk or posedge reset)
   begin
      if (reset)
      begin
         o_FallClk            <= 0;
         o_ps2ControllerData  <= 0;
         r_FilterReg          <= 0;
         r_FilterClk          <= 0;
      end // if statement
      else
      begin
         o_ps2ControllerData  <= i_ps2data;
         o_FallClk            <= 0;
         r_FilterReg          <= {i_ps2clk, r_FilterReg[p_FILTERSIZE-1:1]};
         if (r_FilterReg == 8'b11111111)  // clock level is valid for 8 system clocks
            r_FilterClk       <= 1;
         else if (r_FilterReg == 8'b00000000)
         begin
            r_FilterClk       <= 0;
            if (r_FilterClk == 1)   // if level has changed from the last cycle
            o_FallClk      <= 1;
         end // else if statement
      end // else statement
   end // always block
endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_vga.v                                                                         *
*  Description:     Top level vga module interfacing with ADV7120                                   *
*  Date:            29/11/04                                                                        *
*  Author:           Tim Mamtora                                                                 *
*  Notes:            29/11/04 13:33 First Draft                                                  *
*                                                                                                *
*************************************************************************************************/


module m_vga(clk, reset, i_MemBusy, i_IntensityVal, i_3DColID, i_MColID, i_Override, o_MemStart,
       o_PixelError, o_PixelX, o_PixelY, o_hsync, o_vsync, o_blank, o_RGBout, o_PixelClk, o_FrameSync);

   // parameters - none

   // module inputs
   input                clk;
   input                reset;
   input                i_MemBusy;
   input                i_Override;        // Override for Menu Display
   input [7:0]          i_IntensityVal;
   input [7:0]          i_3DColID;
   input [7:0]          i_MColID;

   // module outputs
   output               o_MemStart;        // Initiate read from Memory Controller
   output               o_PixelError;
   output [8:0]         o_PixelX;
   output [7:0]         o_PixelY;
   output               o_hsync;
   output               o_vsync;
   output               o_blank;           // Blanking signal to the DAC (h and v blanks)
   output [23:0]        o_RGBout;          // Colour output to the DAC
   output               o_PixelClk;        // Clock for the DAC
   output               o_FrameSync;       // Synchroniser to other game modules

   // internal registers

   // internal wires
   wire                 w_PixelBusy;
   wire                 w_LEColIDReg;
   wire                 w_LEIntensityValReg;
   wire [7:0]           w_ColIDin;
   wire [7:0]           w_ColID;
   wire [7:0]           w_IntensityVal;
   wire [23:0]          w_3DCol;
   wire [23:0]          w_RGBCol;
   wire [23:0]          w_MCol;
   wire                 w_vgaBusy;
   wire [15:0]          w_RMultRes;
   wire [15:0]          w_GMultRes;
   wire [15:0]          w_BMultRes;
   wire [7:0]           w_RMult;
   wire [7:0]           w_GMult;
   wire [7:0]           w_BMult;

   // instantiations

   // IntensityVal and ColID registers
   m_Register8    s_ColIDReg        (clk, w_LEColIDReg, w_ColIDin, w_ColID);
   m_Register8    s_IntensityValReg (clk, w_LEIntensityValReg, i_IntensityVal, w_IntensityVal);

   m_vgaTiming    s_vgaTiming       (clk, w_vgaBusy, o_hsync, o_vsync, o_blank, o_PixelX,
                                             o_PixelY, o_FrameSync);
   m_PixelFsm     s_PixelFsm        (clk, reset, i_MemBusy, w_PixelBusy, o_MemStart, w_LEColIDReg,
                                             w_LEIntensityValReg, o_PixelError);

   // Colour palette LUT
   m_colourpalette s_colourpalette  (w_ColID, clk, w_RGBCol);

   // 8-bit multipliers to scale colour values for depth
   m_bmult8x8     s_rmult8x8        (w_RMult, w_IntensityVal, w_RMultRes);
   m_bmult8x8     s_gmult8x8        (w_GMult, w_IntensityVal, w_GMultRes);
   m_bmult8x8     s_bmult8x8        (w_BMult, w_IntensityVal, w_BMultRes);
   // Selector mux for Menu ID or Col ID from video memory
   assign w_ColIDin = i_Override ? i_MColID : i_3DColID;

   // Selector mux for MenuCol or 3DCol to output to the DACs
   assign o_RGBout = i_Override ? w_MCol : w_3DCol;

   // Synchronise VGA with PixelFsm
   assign w_vgaBusy = ~w_PixelBusy;

   // Multiplier inputs
   assign w_RMult = w_RGBCol[23:16];
   assign w_GMult = w_RGBCol[15:8];
   assign w_BMult = w_RGBCol[7:0];

   // Menu Override Colour requires no intensity multiplication
   assign w_MCol  = w_RGBCol;

   // Mulitplier output
   assign w_3DCol = {w_RMultRes[15:8], w_GMultRes[15:8], w_BMultRes[15:8]};

   // Clock for the DAC
   assign o_PixelClk = ~w_PixelBusy;

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_vgaTiming.v                                                                   *
*  Description:     Controls the sync and blanking signals for a 320x240 image display              *
*  Date:            28/11/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           28/11/04 18:12 First Draft                                                      *
*                   02/12/04 19.10 o_PixelX and o_PixelY fixed                                      *
*                                                                                                   *
****************************************************************************************************/



module m_vgaTiming(clk, i_CountEn, i_vgaout, o_hsync, o_vsync, o_blank, o_PixelX, o_PixelY, o_FrameSync);

   // parameters - none

   // module inputs
   input                clk;        // 50MHz system clock
   input                i_CountEn;  // enable signal to time from pixel fsm
   input                i_vgaout;

   // module outputs
   output               o_hsync;
   output               o_vsync;
   output               o_blank;
   output               o_FrameSync;
   output [8:0]         o_PixelX;   // pixel number on current line
   output [7:0]         o_PixelY;   // line number on screen

   // internal registers
   reg                  r_hblank;
   reg                  r_vblank;
   reg                  r_hsync;
   reg                  r_vsync;

   // output registers
   reg                  o_hsync;
   reg                  o_vsync;
   reg                  o_FrameSync;  // active low for 1 clock cycle only
   reg [8:0]            r_hcount;
   reg [9:0]            r_vcount;
   reg [8:0]            o_PixelX;
   reg [7:0]            o_PixelY;

   // module wires
   wire                 w_hsyncon;
   wire                 w_hsyncoff;
   wire                 w_hblankon;
   wire                 w_hreset;
   wire                 w_vsyncon;
   wire                 w_vsyncoff;
   wire                 w_vblankon;
   wire                 w_vreset;

   // horizontal timing
   assign   w_hblankon  = (i_CountEn) & (r_hcount == 320);
   assign   w_hsyncon   = (i_CountEn) & (r_hcount == 326);
   assign   w_hsyncoff  = (i_CountEn) & (r_hcount == 373);
   assign   w_hreset    = (i_CountEn) & (r_hcount == 396);

   // vertical timing
   assign   w_vblankon  = w_hreset & (r_vcount == 479);
   assign   w_vsyncon   = w_hreset & (r_vcount == 492);
   assign   w_vsyncoff  = w_hreset & (r_vcount == 494);
   assign   w_vreset    = w_hreset & (r_vcount == 527);

   // sync and blank generation
   always @(posedge clk)
   begin
      r_hcount    <= (i_CountEn) ? (w_hreset ? 0 : r_hcount + 1) : r_hcount;
      o_PixelX    <= r_hcount;            // output pixel number up to 319
      r_hblank    <= w_hreset    ? 0 : w_hblankon ? 1 : r_hblank;
      r_hsync     <= w_hsyncon   ? 0 : w_hsyncoff ? 1 : o_hsync;  // hsync is active low

      r_vcount    <= w_hreset    ? (w_vreset ? 0 : r_vcount + 1) : r_vcount;
      o_PixelY    <= r_vcount[8:1];       // output line number up to 239
      r_vblank    <= w_vreset    ? 0 : w_vblankon ? 1 : r_vblank;
      r_vsync     <= w_vsyncon   ? 0 : w_vsyncoff ? 1 : o_vsync;  // vsync is active low

      o_hsync     <= i_vgaout ? r_hsync : o_hsync;   // output given pixel after memory lookup
      o_vsync     <= i_vgaout ? r_vsync : o_vsync;

      o_FrameSync <= ((r_hcount == 0) & (r_vcount == 0)) ? 1 : 0;
   end // always statement

    assign o_blank = ~(r_hblank | r_vblank);
endmodule

/*********************************************************************************************************************
*                                                                                                                   *
*   Filename:       m_3DAICCalculateFSM.v                                                                           *
*   Description:    Controls the generation of information for the AIC RAM.                                         *
*   Date:           03/12/2004 21:33                                                                                *
*   Author:         Robert Cassels                                                                                  *
*   Notes:          None                                                                                            *
*                                                                                                                   *
*********************************************************************************************************************/

module m_3DAICCalculateFSM(clk, reset, i_Start, o_Select, o_WriteEnable, o_Busy, o_Load);

    // Parameters

    parameter p_SELECT_DIV              = 6;
    parameter p_SELECT_ATAN             = 0;
    parameter p_SELECT_SQRT             = 5;

    parameter p_STATE_IDLE              = 00;
    parameter p_STATE_WAITPROP          = 01;
    parameter p_STATE_LOOUPDIV          = 02;
    parameter p_STATE_WAITDIV           = 03;
    parameter p_STATE_LOOKUPSQRT        = 04;
    parameter p_STATE_WAITSQRT1         = 05;
    parameter p_STATE_WAITSQRT2         = 06;
    parameter p_STATE_WAITSQRT3         = 07;
    parameter p_STATE_WAITSQRT4         = 08;
    parameter p_STATE_WAITSQRT5         = 09;
    parameter p_STATE_LOOKUPATAN        = 10;
    parameter p_STATE_WAITATAN1         = 11;
    parameter p_STATE_WAITATAN2         = 12;
    parameter p_STATE_LOADRAM           = 13;

    // Inputs

    input                               i_Start;
    input                               clk;
    input                               reset;

    // Outputs

    output [2:0]                        o_Select;
    output                              o_WriteEnable;
    output                              o_Load;
    output                              o_Busy;

    // Bidirectionals

    // Internal Registers

    reg [3:0]                           r_State;

    // Output Registers

    reg [2:0]                           o_Select;
    reg                                 o_Load;
    reg                                 o_WriteEnable;
    reg                                 o_Busy;

    // Synchronous Block

    always @ (posedge clk)
    begin

        // Update state

        case (r_State)
            p_STATE_IDLE:       r_State = reset ? p_STATE_IDLE : i_Start ? p_STATE_WAITPROP : p_STATE_IDLE;
            p_STATE_LOADRAM:    r_State = p_STATE_IDLE;
            default:            r_State = reset ? p_STATE_IDLE : r_State + 1;
        endcase

        case (r_State)
            p_STATE_LOOUPDIV:   o_Select = p_SELECT_DIV;
            p_STATE_LOOKUPSQRT: o_Select = p_SELECT_SQRT;
            p_STATE_LOOKUPATAN: o_Select = p_SELECT_ATAN;
            default:            o_Select = 3'bXXX;
        endcase

        case (r_State)
            p_STATE_LOOUPDIV:   o_Load = 1;
            p_STATE_LOOKUPSQRT: o_Load = 1;
            p_STATE_LOOKUPATAN: o_Load = 1;
            default:            o_Load = 0;
        endcase

        o_WriteEnable   = (r_State == p_STATE_LOADRAM) ? 1 : 0;
        o_Busy          = ((r_State != p_STATE_IDLE) && (r_State != p_STATE_LOADRAM)) ? 1 : 0;
    end

endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_3DRenderer.v                                                                               *
*   Description:    Casts rays to find wall intersections.                                                       *
*   Date:           21/11/2004 12:38                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          21/11/2004 18:02 - First compiled successfully                                               *
*                   22/11/2004 00:34 - Simulation complete - Most bugs removed                                   *
*                                                                                                                *
*****************************************************************************************************************/
```

```
module m_3DRayCaster(clk, i_IncrementX, i_IncrementY, i_PositionX, i_PositionY,
                        i_LoadNew, i_IncrementNow, o_WallX, o_WallY, o_Side);

    // Parameters

    parameter p_WIDTH_INCREMENT         = (12 - 1);
    parameter p_WIDTH_POSITION          = (12 - 1);
    parameter p_WIDTH_DIFFERENCE        = (12 - 1);

    // Inputs

    input [p_WIDTH_INCREMENT:0]         i_IncrementX;
    input [p_WIDTH_INCREMENT:0]         i_IncrementY;

    input [p_WIDTH_POSITION:0]          i_PositionX;
    input [p_WIDTH_POSITION:0]          i_PositionY;

    input                               i_LoadNew;
    input                               i_IncrementNow;

    input                               clk;

    // Outputs

    output [p_WIDTH_POSITION:0]         o_WallX;
    output [p_WIDTH_POSITION:0]         o_WallY;

    output                              o_Side;

    // Bidirectionals

    // Wires

    wire [p_WIDTH_POSITION:0]           w_StartX;
    wire [p_WIDTH_POSITION:0]           w_StartY;

    wire                                w_Inc1;
    wire                                w_Inc2;

    wire                                w_XValid;
    wire                                w_YValid;

    wire [p_WIDTH_POSITION:0]           w_CurrentX1;
    wire [p_WIDTH_POSITION:0]           w_CurrentY1;
    wire [p_WIDTH_POSITION:0]           w_CurrentX2;
    wire [p_WIDTH_POSITION:0]           w_CurrentY2;

    wire [p_WIDTH_POSITION:0]           w_XAIn;
    wire [p_WIDTH_POSITION:0]           w_XBIn;
    wire [p_WIDTH_POSITION:0]           w_YAIn;
    wire [p_WIDTH_POSITION:0]           w_YBIn;

    wire [p_WIDTH_DIFFERENCE:0]         w_Difference;
    wire                                w_Carry;
    wire                                w_Choice;

    // Instantiation

    m_InitialValueCalculator initvalcalc(.i_XIncrement(i_IncrementX), .i_YIncrement(i_IncrementY),
                                  .i_PositionX(i_PositionX), .i_PositionY(i_PositionY),
                                  .o_InitialPositionX(w_StartX), .o_InitialPositionY(w_StartY),
                                  .o_XValid(w_XValid), .o_YValid(w_YValid));

    m_ParametricIncrementRegister para_reg_x(.clk(clk), .i_InitialCoordinate(w_StartX),
                                  .i_IncrementValue(i_IncrementX), .i_LoadInitialValue(i_LoadNew),
                                  .i_Increment(w_Inc1), .o_FinalCoordinate(w_CurrentX1));

    m_SingleIncrementRegister sing_reg_y(.clk(clk), .i_InitialCoordinate(i_PositionY[11:6]),
                                  .i_DirectionalIncrement(i_IncrementX[p_WIDTH_INCREMENT]),
                                  .i_LoadInitialValue(i_LoadNew), .i_Increment(w_Inc1),
                                  .o_AlignedCoordinate(w_CurrentY1));
    m_ParametricIncrementRegister para_reg_y(.clk(clk), .i_InitialCoordinate(w_StartY),
                                   .i_IncrementValue(i_IncrementY), .i_LoadInitialValue(i_LoadNew),
                                   .i_Increment(w_Inc2), .o_FinalCoordinate(w_CurrentY2));

    m_SingleIncrementRegister sing_reg_x(.clk(clk), .i_InitialCoordinate(i_PositionX[11:6]),
                                   .i_DirectionalIncrement(i_IncrementY[p_WIDTH_INCREMENT]),
                                   .i_LoadInitialValue(i_LoadNew), .i_Increment(w_Inc2),
                                   .o_AlignedCoordinate(w_CurrentX2));

    // Calculate which wall is closer

    // Setup the correct signs

    assign w_XAIn = i_IncrementX[p_WIDTH_INCREMENT] ? w_CurrentX1 : w_CurrentX2;
    assign w_XBIn = i_IncrementX[p_WIDTH_INCREMENT] ? w_CurrentX2 : w_CurrentX1;
    assign w_YAIn = i_IncrementY[p_WIDTH_INCREMENT] ? w_CurrentY1 : w_CurrentY2;
    assign w_YBIn = i_IncrementY[p_WIDTH_INCREMENT] ? w_CurrentY2 : w_CurrentY1;

    // Calculate the difference

    assign {w_Carry, w_Difference} = w_XAIn + w_YAIn - w_XBIn - w_YBIn;

    // Compensate for invalid inputs

    assign w_Choice = (w_XValid) & ((~w_YValid) | (~w_Carry));

    // Increment the correct counters

    assign w_Inc1 = i_IncrementNow ?  w_Choice : 0;
    assign w_Inc2 = i_IncrementNow ? ~w_Choice : 0;

    // Output the correct coordinates

    assign o_WallX = w_Choice ? w_CurrentX1 : w_CurrentX2;
    assign o_WallY = w_Choice ? w_CurrentY1 : w_CurrentY2;

    // Output if the wall is to the "side" or at the "top"

    assign o_Side = ~w_Choice;
endmodule

/*****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_3DRenderer.v                                                                              *
*   Description:    Renders the 3D image.                                                                       *
*   Date:           11/11/2004 11:20                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          04/12/2004 16:36 - First successful compile.                                                *
*                                                                                                               *
*****************************************************************************************************************/

module m_3DRenderer(clk, reset, i_PlayerPosX, i_PlayerPosY, i_PlayerAngle, i_AICPositionX, i_AICPositionY,
                    i_PosUpdateIsSide, i_PosUpdateWallX, i_PosUpdateWallY, i_MapID, i_ROMData, i_PosUpdateFetch,
                    i_StartFrame, o_ROMAddress, o_Busy, o_VidMemWrite, o_ScreenY, o_ScreenX, o_AICID, o_ColourID,
                    o_Intensity, o_PosUpdateIsWall, o_DirBitX, o_DirBitY);

    // Parameters

    parameter p_WIDTH_THETA             = (11 - 1);
    parameter p_WIDTH_ADDRESS           = (19 - 1);
    parameter p_WIDTH_DATA              = (8  - 1);
    parameter p_WIDTH_POSITION          = (12 - 1);
    parameter p_WIDTH_TEXTUREID         = (4  - 1);
    parameter p_WIDTH_COLOURID          = (8  - 1);
    parameter p_WIDTH_MAPID             = (4  - 1);
    parameter p_WIDTH_INTENSITY         = (8  - 1);
    parameter p_WIDTH_DEPTH             = (12 - 1);
    parameter p_WIDTH_WALLCOORDINATE    = (6  - 1);
    parameter p_WIDTH_TEXTURECOORDINATE = (6  - 1);
    parameter p_WIDTH_SCREENX           = (9  - 1);
    parameter p_WIDTH_SCREENY           = (8  - 1);
    parameter p_WIDTH_AICID             = (3  - 1);
    parameter p_WIDTH_HEIGHT            = (8  - 1);
    parameter p_WIDTH_HSCALE            = (8  - 1);
    parameter p_WIDTH_DIVCONST          = (12 - 1);
    parameter p_WIDTH_RATIO             = (12 - 1);
    parameter p_WIDTH_ENCODEDSQUARE     = (12 - 1);
    parameter p_WIDTH_USELESS1          = (12 - 1);
    parameter p_WIDTH_ROMLOOKUP         = (3  - 1);
    parameter p_WIDTH_DIFFERENCE        = (13 - 1);
    parameter p_WIDTH_DENOMINATOR       = (12 - 1);

    // Inputs

    input [p_WIDTH_POSITION:0]          i_PlayerPosX;
    input [p_WIDTH_POSITION:0]          i_PlayerPosY;
    input [p_WIDTH_THETA:0]             i_PlayerAngle;

    input [p_WIDTH_POSITION:0]          i_AICPositionX;
    input [p_WIDTH_POSITION:0]          i_AICPositionY;

    input                               i_PosUpdateIsSide;
    input [p_WIDTH_WALLCOORDINATE:0]    i_PosUpdateWallX;
    input [p_WIDTH_WALLCOORDINATE:0]    i_PosUpdateWallY;
    input                               i_PosUpdateFetch;
    input [p_WIDTH_MAPID:0]             i_MapID;

    input [p_WIDTH_DATA:0]              i_ROMData;

    input                               i_StartFrame;

    input                               clk;
    input                               reset;

    // Outputs

    output [p_WIDTH_ADDRESS:0]          o_ROMAddress;

    output [p_WIDTH_SCREENY:0]          o_ScreenY;
    output [p_WIDTH_SCREENX:0]          o_ScreenX;

    output [p_WIDTH_AICID:0]            o_AICID;

    output [p_WIDTH_COLOURID:0]         o_ColourID;
    output [p_WIDTH_INTENSITY:0]        o_Intensity;

    output                              o_PosUpdateIsWall;
    output                              o_DirBitX;
    output                              o_DirBitY;

    output                              o_Busy;

    output                              o_VidMemWrite;

    // Bidirectionals

    // Wires

    wire [p_WIDTH_SCREENY:0]            w_ScreenY;
    wire [p_WIDTH_SCREENX:0]            w_ScreenX;

    wire [p_WIDTH_DIFFERENCE:0]         w_DiffX;
    wire [p_WIDTH_DIFFERENCE:0]         w_DiffY;

    wire [p_WIDTH_POSITION:0]           w_MuxedPositionX;
    wire [p_WIDTH_POSITION:0]           w_MuxedPositionY;
    wire [p_WIDTH_POSITION:0]           w_WallX;
    wire [p_WIDTH_POSITION:0]           w_WallXDelayed;
    wire [p_WIDTH_POSITION:0]           w_WallY;
    wire [p_WIDTH_POSITION:0]           w_WallYDelayed;
    wire [p_WIDTH_POSITION:0]           w_IncrementX;
    wire [p_WIDTH_POSITION:0]           w_IncrementY;
    wire [p_WIDTH_POSITION:0]           w_Numerator;
    wire [p_WIDTH_POSITION:0]           w_DiffYAbs;
    wire [p_WIDTH_POSITION:0]           w_DiffXAbs;

    wire [p_WIDTH_THETA:0]              w_AICTheta;
    wire [p_WIDTH_THETA:0]              w_ArcTan;
    wire [p_WIDTH_THETA:0]              w_Theta;
    wire [p_WIDTH_THETA:0]              w_CurrentAngle;

    wire [p_WIDTH_WALLCOORDINATE:0]     w_MuxedMapX;
    wire [p_WIDTH_WALLCOORDINATE:0]     w_MuxedMapY;

    wire [p_WIDTH_TEXTUREID:0]          w_MuxedTexID;
    wire [p_WIDTH_TEXTUREID:0]          w_WallTexID;

    wire [p_WIDTH_TEXTURECOORDINATE:0]  w_AICTexU;
    wire [p_WIDTH_TEXTURECOORDINATE:0]  w_MuxedTexU;
    wire [p_WIDTH_TEXTURECOORDINATE:0]  w_TextureV;

    wire [p_WIDTH_DEPTH:0]              w_AICDepth;
    wire [p_WIDTH_DEPTH:0]              w_ROMDepth;
    wire [p_WIDTH_DEPTH:0]              w_MuxedDepth;

    wire [p_WIDTH_COLOURID:0]           w_TexColour;
    wire [p_WIDTH_COLOURID:0]           w_FinColourID;

    wire [p_WIDTH_HSCALE:0]             w_AICHScale;
    wire [p_WIDTH_HSCALE:0]             w_MuxedHScale;
    wire [p_WIDTH_HSCALE:0]             w_ROMHScale;

    wire [p_WIDTH_HEIGHT:0]             w_AICHeight;
    wire [p_WIDTH_HEIGHT:0]             w_MuxedHeight;
    wire [p_WIDTH_HEIGHT:0]             w_ROMHeight;

    wire [p_WIDTH_INTENSITY:0]          w_AICIntensity;
    wire [p_WIDTH_INTENSITY:0]          w_ROMIntensity;
    wire [p_WIDTH_INTENSITY:0]          w_MuxedIntensity;
    wire [p_WIDTH_INTENSITY:0]          w_FinIntensity;

    wire [p_WIDTH_AICID:0]              w_CurAIC;

    wire [p_WIDTH_DIVCONST:0]           w_DivConst;

    wire [p_WIDTH_RATIO:0]              w_Ratio;

    wire [p_WIDTH_ENCODEDSQUARE:0]      w_ROMDepthSqaured;

    wire [p_WIDTH_USELESS1:0]           w_Useless1;

    wire [p_WIDTH_ROMLOOKUP:0]          w_LookupSelect;

    wire [p_WIDTH_DENOMINATOR:0]        w_Denominator;

    wire                                w_AICInfoStore;
    wire                                w_CasterIncrement;
    wire                                w_DiffXBigger;
    wire                                w_IsSide;
    wire                                w_IsSideDelayed;
    wire                                w_LoadCaster;
    wire                                w_Lookup;
    wire                                w_MakeAIC;
    wire                                w_MuxedSideSelect;
    wire                                w_RenderRunning;
    wire                                w_TexLookupUValid;
    wire                                w_UseAICTex;
    wire                                w_VTexValid;
    wire                                w_WallDelayLoad;
    wire                                w_DirectionX;
    wire                                w_DirectionY;
    wire                                w_TryAICPixel;
    wire                                w_LoadValid;
    wire                                w_ColValid;
    wire                                w_ColValidDelayed;
    wire                                w_ForcePixLoad;
    wire                                w_CmpPixLoad;
    wire                                w_LoadHold;
    wire                                w_LoadCoord;

    // Instantiation

    m_3DRayCaster caster(.clk(clk),                               // The clock (global)
                        .i_IncrementX(w_IncrementX),              // The increments due to the current user
                        .i_IncrementY(w_IncrementY),              //    angle (from ROM COntroller)
                        .i_PositionX(i_PlayerPosX),               // The position of the player
                        .i_PositionY(i_PlayerPosY),               //  (an input this module)
                        .i_LoadNew(w_LoadCaster),                 // Commands the caster to load new values (from FSM)
                        .i_IncrementNow(w_CasterIncrement),       // Commands the caster to compute the next value (FSM)
                        .o_WallX(w_WallX),                        // The position of the current wall intersection to try
                        .o_WallY(w_WallY),                        //  (goes to the ROM Controller)
                        .o_Side(w_IsSide));                       // The orientation of the current wall (to ROM Controller)

    m_aicsram49x8 AICinfo(.clk(clk),
                        .addr(w_CurAIC),
                        .we(w_AICInfoStore),
                        .din( {w_ROMHeight, w_ROMHScale, w_Theta,    w_ROMDepth, w_ROMIntensity, w_DirectionX, w_DirectionY}),
                        .dout({w_AICHeight, w_AICHScale, w_AICTheta, w_AICDepth, w_AICIntensity, o_DirBitX,    o_DirBitY}));

    m_PositionToDepthSquared p2ds(.i_PlayerX(i_PlayerPosX),       // The position of the player
                        .i_PlayerY(i_PlayerPosY),                 //  (an input to this module)
                        .i_CharacterX(w_MuxedPositionX),          // The position of the current AIC character or wall to get
                        .i_CharacterY(w_MuxedPositionY),          //  distance squared to (from either AIC Pos or Delayed Wall)
                        .o_DifferenceX(w_DiffX),                  // The difference between the two positions, this is used to
                        .o_DifferenceY(w_DiffY),                  //  compute the angle between them (using the divider)
                        .o_DepthSquared(w_ROMDepthSqaured));      // The encoded depth squared (to ROM COntroller for lookup)

    m_ROMController ROMCont(.clk(clk),                            // The clock (global)
                        .reset(reset),                            // The global reset
                        .i_Angle(w_CurrentAngle),                 // The angle of the current pixel column
                        .i_SquareValue(w_ROMDepthSqaured),        // The encoded square of the distance to the wall or characte
                        .i_MapID(i_MapID),                        // The current Map ID
                        .i_MapX(w_MuxedMapX),                     // The current position in the map to check (either from
                        .i_MapY(w_MuxedMapY),                     //  position update module or the ray caster)
                        .i_MapBlockSideSelect(w_MuxedSideSelect), // Selects whether the side or top of a block is checked
                        .i_TextureID(w_MuxedTexID),               // The ID of the texture to lookup
                        .i_TextureU(w_MuxedTexU),                 // The texture coordinates to lookup (from m_ScreenYToTextrue
                        .i_TextureV(w_TextureV),                  //  and m_ScreenXToTextrueU or RayCaster directly)
                        .i_Denominator(w_Denominator),            // The value to find the division constant for (from p2ds)
                        .i_OppAdjRatio(w_Ratio),                  // The ratio to do the arctan of (from the division)
                        .i_ROMData(i_ROMData),                    // The inbound data bus
                        .i_ItemSelect(w_LookupSelect),            // The new thing to look up (from FSM)
                        .i_Load(w_Lookup),                        // The command to do the lookup (from FSM)
                        .o_AngleXChange(w_IncrementX),            // The increments for the ray caster
                        .o_AngleYChange(w_IncrementY),            //  (to ray caster)
                        .o_SquareRootValue(w_ROMDepth),           // The depth for the X to U conversion
                        .o_IntensityValue(w_ROMIntensity),        // The intensity for the video memory (possibly via the FSM)
                        .o_ScreenHeight(w_ROMHeight),             // The height of a texture (eg a wall) on screen
                        .o_HeightScale(w_ROMHScale),              // The sclae by which to change the screen height
                        .o_MapTextureID(w_WallTexID),             // The texture ID found at the current map location
                        .o_ColourID(w_TexColour),                 // The colour found in the current texture
                        .o_DivisionConstant(w_DivConst),          // The constant for division correspoinding to w_Denominator
                        .o_Arctangent(w_ArcTan),                  // The arctangent of w_Ratio
                        .o_ROMAddress(o_ROMAddress));             // The current ROM address for lookup

    m_ScreenYToTextrueV scrntotex(.i_HeightScale(w_MuxedHScale),  // The amount by which to scale the height (ROM/AIC RAM)
                        .i_Height(w_MuxedHeight),                 // The height of the texture on the screen (ROM/AIC RAM)
                        .i_ScreenY(w_ScreenY),                    // The current pixel to draw (FSM)
                        .o_TextureV(w_TextureV),                  // The texture coordinate that should be read
                        .o_CoordinateValid(w_VTexValid));         // Indicates whether the coordinate is valid (out of bounds)

    m_WallPositionDelay walldelay(.clk(clk),                      // The global clock
                        .i_Load(w_WallDelayLoad),                 // Commands the delay to load a new value (from FSM)
                        .i_WallPositionX(w_WallX),                // The values from the ray caster
                        .i_WallPositionY(w_WallY),                //
                        .i_IsSide(w_IsSide),                      //
                        .o_WallPositionX(w_WallXDelayed),         // The delayed values
```

```
                        .o_WallPositionY(w_WallYDelayed),         //
                        .o_IsSide(w_IsSideDelayed));              //

    m_ScreenXToTextrueU aicscrntotex(.i_Depth(w_AICDepth),        // The depth from the player of the AIC
                        .i_ScreenTheta(w_CurrentAngle),           // The current pixel column angle
                        .i_DirectionTheta(w_AICTheta),            // The angle of the AIC
                        .o_TextureU(w_AICTexU),                   // The resulting texture U coordinate
                        .o_CoordinateValid(w_TexLookupUValid));   // Whether the result is within bounds (valid or not)

    m_multiplier12sx12s mult(.a(w_DivConst),                      // The dividing constant
                        .b(w_Numerator),                          // The the number to divide
                        .o({w_Ratio,w_Useless1}));                // The result of the division

    m_PixelRegister pixreg(.clk(clk),                             // The global clock
                        .i_ColourID(w_TexColour),                 //
                        .i_Depth(w_MuxedDepth),                   //
                        .i_Intensity(w_MuxedIntensity),           //
                        .i_Valid(w_ColValidDelayed),              //
                        .i_ForceLoad(w_ForcePixLoad),             //
                        .i_CompareLoad(w_CmpPixLoad),             //
                        .o_ColourID(w_FinColourID),               //
                        .o_Intensity(w_FinIntensity),             //
                        .i_PixelY(w_ScreenY));                    //

    m_OneBitDelay onebitreg(.clk(clk),                            // The global clock
                        .i_Value(w_ColValid),                     //
                        .i_Load(w_LoadValid),                     //
                        .o_Value(w_ColValidDelayed));             //

    m_3DRendererFSM fsm(.clk(clk),                                //
                        .reset(reset),                            //
                        .i_StartFrame(i_StartFrame),              //
                        .i_IsWall(o_PosUpdateIsWall),             //
                        .i_PosUpdt(i_PosUpdateFetch),             //
                        .o_Busy(o_Busy),                          //
                        .o_WriteEnable(w_AICInfoStore),           //
                        .o_ForcePixLoad(w_ForcePixLoad),          //
                        .o_CmpPixLoad(w_CmpPixLoad),              //
                        .o_AICID(o_AICID),                        //
                        .o_TryAICPixel(w_TryAICPixel),            //
                        .o_UseAICTex(w_UseAICTex),                //
                        .o_LoadValid(w_LoadValid),                //
                        .o_LoadHold(w_LoadHold),                  //
                        .o_LoadCoord(w_LoadCoord),                //
                        .o_WriteToMem(o_VidMemWrite),             //
                        .o_ScreenY(w_ScreenY),                    //
                        .o_ScreenX(w_ScreenX),                    //
                        .o_ROMSelect(w_LookupSelect),             //
                        .o_ROMLoad(w_Lookup),                     //
                        .o_LoadCaster(w_LoadCaster),              //
                        .o_WallDelayLoad(w_WallDelayLoad),        //
                        .o_CasterIncrement(w_CasterIncrement),    //
                        .o_MakeAIC(w_MakeAIC));                   //

    m_HoldReg hreg(.clk(clk),
                        .i_InfoLoad(w_LoadHold),
                        .i_CoordLoad(w_LoadCoord),
                        .i_Colour(w_FinColourID),
                        .i_ScreenX(w_ScreenX),
                        .i_ScreenY(w_ScreenY),
                        .i_Intensity(w_FinIntensity),
                        .o_ScreenX(o_ScreenX),
                        .o_ScreenY(o_ScreenY),
                        .o_ColourID(o_ColourID),
                        .o_Intensity(o_Intensity));

//assign o_ColourID = o_ScreenX[7:0];
//assign o_Intensity = o_ScreenY[7:0];

    m_cmpl12 comparesize(.A(w_DiffYAbs),
                        .B(w_DiffXAbs),
                        .A_LT_B(w_DiffXBigger));
    // Assignments

    // Input to "Position To Depth Squared" module

    assign w_MuxedPositionX     = w_MakeAIC ? i_AICPositionX : w_WallXDelayed;
    assign w_MuxedPositionY     = w_MakeAIC ? i_AICPositionY : w_WallYDelayed;

    // Input to ROM Controller

    assign w_MuxedMapX          = w_RenderRunning ? w_WallX[5:0]    : i_PosUpdateWallX;
    assign w_MuxedMapY          = w_RenderRunning ? w_WallY[5:0]    : i_PosUpdateWallY;
    assign w_MuxedSideSelect    = w_RenderRunning ? w_IsSide        : i_PosUpdateIsSide;

    // Input to m_ScreenYToTextrueV

    assign w_MuxedHScale        = w_UseAICTex ? w_AICHScale : w_ROMHScale;
    assign w_MuxedHeight        = w_UseAICTex ? w_AICHeight : w_ROMHeight;

    // Input to ROM Controller

    assign w_MuxedTexID         = w_UseAICTex ? {1'b1, w_CurAIC} : w_WallTexID;
    assign w_MuxedTexU          = w_UseAICTex ? w_AICTexU : w_IsSideDelayed ? w_WallYDelayed[5:0] : w_WallXDelayed[5:0];

    // The current vertical pixel column's angle

    assign w_CurrentAngle       = i_PlayerAngle - {2'b00, w_ScreenX};

    // The Numerator and denominator to enter the division multiplier and ROM Controller

    assign w_DiffYAbs           = w_DiffY[12]   ? -w_DiffY[p_WIDTH_POSITION:0] : w_DiffY[p_WIDTH_POSITION:0];
    assign w_DiffXAbs           = w_DiffX[12]   ? -w_DiffX[p_WIDTH_POSITION:0] : w_DiffX[p_WIDTH_POSITION:0];

    assign w_Denominator        = w_DiffXBigger ? w_DiffYAbs : w_DiffXAbs;
    assign w_Numerator          = w_DiffXBigger ? w_DiffXAbs : w_DiffYAbs;

    // The angle of the current AIC relative to the player (input to the AIC RAM)

    assign w_Theta              = (w_DiffXBigger ? (w_DiffY[12] ? 1536 : 512) : (w_DiffX[12] ? 1024 : 0)) +
                                    ((w_DiffXBigger ^ w_DiffY[12] ^ w_DiffX[12]) ? -w_ArcTan : w_ArcTan);

    // Assign inputs to the pixel register

    assign w_MuxedDepth         = w_TryAICPixel ? w_AICDepth        : w_ROMDepth;
    assign w_MuxedIntensity     = w_TryAICPixel ? w_AICIntensity    : w_ROMIntensity;

    //

    assign w_ColValid           = w_VTexValid & (w_UseAICTex ? 0 : 1); // alive & w_TexLookupUValid

    // Assign the position update outputs

    assign o_PosUpdateIsWall    = (w_WallTexID == 0) ? 0 : 1;

    assign w_RenderRunning      = o_Busy;
    assign w_CurAIC             = o_AICID;

endmodule

/*************************************************************************************************************
*                                                                                                            *
*   Filename:       m_3DRendererFSM.v                                                                        *
*   Description:    The instantiations of all 3D modules                                                     *
*   Date:           27/11/2004 16:09                                                                         *
*   Author:         Robert Cassels                                                                           *
*   Notes:          None                                                                                     *
*                                                                                                            *
*************************************************************************************************************/

module m_3DRendererFSM(clk, reset, i_StartFrame, i_IsWall, i_PosUpdt, o_Busy, o_WriteEnable, o_ForcePixLoad,
                    o_CmpPixLoad, o_AICID, o_TryAICPixel, o_UseAICTex, o_LoadValid, o_LoadHold, o_LoadCoord,
                    o_WriteToMem, o_ScreenY, o_ScreenX, o_ROMSelect, o_ROMLoad, o_LoadCaster, o_WallDelayLoad,
                    o_CasterIncrement, o_MakeAIC, w_DrawBusy);

    output w_DrawBusy;

    // Parameters

    parameter p_WIDTH_SCREENY           = (8  - 1);
    parameter p_WIDTH_SCREENX           = (9  - 1);
    parameter p_WIDTH_SELECT            = (3  - 1);
    parameter p_WIDTH_AICID             = (3  - 1);

    parameter p_STATE_AICSTART          = 0;
    parameter p_STATE_AICWAIT           = 1;
    parameter p_STATE_AICLOOP           = 2;
    parameter p_STATE_DRAWSTART         = 3;
    parameter p_STATE_DRAWWAIT          = 4;
    parameter p_STATE_DRAWLOOP          = 5;
    parameter p_STATE_IDLE              = 6;

    // Inputs

    input                               i_StartFrame;
    input                               i_IsWall;
    input                               i_PosUpdt;

    input                               clk;
    input                               reset;

    // Outputs

    output                              o_Busy;
    output                              o_WriteEnable;
    output                              o_ForcePixLoad;
    output                              o_CmpPixLoad;
    output [p_WIDTH_AICID:0]            o_AICID;
    output                              o_TryAICPixel;
    output                              o_UseAICTex;
    output                              o_LoadValid;
    output                              o_LoadHold;
    output                              o_LoadCoord;
    output                              o_WriteToMem;
    output [p_WIDTH_SCREENY:0]          o_ScreenY;
    output [p_WIDTH_SCREENX:0]          o_ScreenX;
    output [p_WIDTH_SELECT:0]           o_ROMSelect;
    output                              o_ROMLoad;
    output                              o_LoadCaster;
    output                              o_WallDelayLoad;
    output                              o_CasterIncrement;
    output                              o_MakeAIC;


    // Internal Registers

    reg [3:0]                           r_State;
    reg [8:0]                           r_Count;
    reg                                 r_AICStart;
    reg                                 r_ColStart;

    // Output Registers

    reg                                 o_Busy;
    reg [p_WIDTH_SELECT:0]              o_ROMSelect;
    reg                                 o_ROMLoad;
    reg [p_WIDTH_AICID:0]               o_AICID;
    reg                                 o_MakeAIC;

    // Wires

    wire [p_WIDTH_SELECT:0]             w_AICSelect;
    wire [p_WIDTH_SELECT:0]             w_ROMSelect;
    wire [p_WIDTH_AICID:0]              w_AICID;
    wire                                w_ROMLoad;
    wire                                w_AICLoad;
    wire                                w_AICBusy;
    wire                                w_DrawBusy;

    // Instantiations

    m_3DAICCalculateFSM AICCalc(.clk(clk),
                                .reset(reset),
                                .i_Start(r_AICStart),
                                .o_Select(w_AICSelect),
                                .o_WriteEnable(o_WriteEnable),
                                .o_Busy(w_AICBusy),
                                .o_Load(w_AICLoad));

    m_ColumnRenderFSM ColRend(.clk(clk),
                                .reset(reset),
                                .i_Start(r_ColStart),
                                .i_IsWall(i_IsWall),
                                .o_ForcePixLoad(o_ForcePixLoad),
                                .o_CmpPixLoad(o_CmpPixLoad),
                                .o_AICID(w_AICID),
                                .o_TryAICPixel(o_TryAICPixel),
                                .o_UseAICTex(o_UseAICTex),
                                .o_LoadValid(o_LoadValid),
                                .o_LoadHold(o_LoadHold),
                                .o_LoadCoord(o_LoadCoord),
                                .o_WriteToMem(o_WriteToMem),
                                .o_ScreenY(o_ScreenY),
                                .o_ROMSelect(w_ROMSelect),
                                .o_ROMLoad(w_ROMLoad),
                                .o_Busy(w_DrawBusy),
                                .o_LoadCaster(o_LoadCaster),
                                .o_WallDelayLoad(o_WallDelayLoad),
                                .o_CasterIncrement(o_CasterIncrement));

    // Sychonous Block

    always @ (posedge clk)
    begin

        // Do main states

        case (r_State)
            p_STATE_AICSTART:   r_State = reset ? p_STATE_IDLE : p_STATE_AICWAIT;
            p_STATE_AICWAIT:    r_State = reset ? p_STATE_IDLE : w_AICBusy ? p_STATE_AICWAIT : p_STATE_AICLOOP;
            p_STATE_AICLOOP:    r_State = reset ? p_STATE_IDLE : (r_Count == 8) ? p_STATE_DRAWSTART : p_STATE_AICWAIT;
            p_STATE_DRAWSTART:  r_State = reset ? p_STATE_IDLE : p_STATE_DRAWWAIT;
            p_STATE_DRAWWAIT:   r_State = reset ? p_STATE_IDLE : w_DrawBusy ? p_STATE_DRAWWAIT : p_STATE_DRAWLOOP;
            p_STATE_DRAWLOOP:   r_State = reset ? p_STATE_IDLE : (r_Count == 320) ? p_STATE_IDLE : p_STATE_DRAWWAIT;
            p_STATE_IDLE:       r_State = reset ? p_STATE_IDLE : i_StartFrame ? p_STATE_AICSTART : p_STATE_IDLE;
            default:            r_State = p_STATE_IDLE;
        endcase

        // Set the busy signal for all but idle

        o_Busy      = (r_State == p_STATE_IDLE) ? 0 : 1;

        // Count when looping

        r_Count     = ((r_State == p_STATE_AICLOOP )  || (r_State == p_STATE_DRAWLOOP )) ? (r_Count + 1) :
                      (r_State == p_STATE_AICSTART) ? 1 : (r_State == p_STATE_DRAWSTART) ? 0 : r_Count;

        // Start the FSMs

        r_AICStart  = ((r_State == p_STATE_AICSTART ) || ((r_State == p_STATE_AICLOOP ) && (r_Count != 8  ))) ? 1 : 0;
        r_ColStart  = ((r_State == p_STATE_DRAWSTART) || ((r_State == p_STATE_DRAWLOOP) && (r_Count != 320))) ? 1 : 0;

        o_ROMSelect = ((r_State == p_STATE_IDLE) & i_PosUpdt) ? 2 :
                        (r_State == p_STATE_AICWAIT) ? w_AICSelect : w_ROMSelect;
        o_ROMLoad   = ((r_State == p_STATE_IDLE) & i_PosUpdt) ? 1 :
                        (r_State == p_STATE_AICWAIT) ? w_AICLoad   : w_ROMLoad;
        o_AICID     = (r_State == p_STATE_AICWAIT) ? r_Count[2:0] : w_AICID;
```

```
        case (r_State)
            p_STATE_AICSTART:   o_MakeAIC = 1;
            p_STATE_AICWAIT:    o_MakeAIC = 1;
            p_STATE_AICLOOP:    o_MakeAIC = 1;
            default:            o_MakeAIC = 0;
        endcase
    end

    assign o_ScreenX = r_Count;
endmodule

/*****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_ColumnRenderFSM.v                                                                         *
*   Description:    Renders a whole column of pixels.                                                           *
*   Date:           03/12/2004 23:54                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          None                                                                                        *
*                                                                                                               *
*****************************************************************************************************************/

module m_ColumnRenderFSM(clk, reset, i_Start, i_IsWall, o_ForcePixLoad, o_CmpPixLoad, o_AICID, o_TryAICPixel,
                    o_UseAICTex, o_LoadValid, o_LoadHold, o_LoadCoord, o_WriteToMem, o_ScreenY, o_ROMSelect,
                    o_ROMLoad, o_Busy, o_LoadCaster, o_WallDelayLoad, o_CasterIncrement);

    // Parameters

    parameter p_WIDTH_AICID             = (3  - 1);
    parameter p_WIDTH_SCREENY           = (8  - 1);
    parameter p_WIDTH_SELECT            = (3  - 1);

    parameter p_STATE_IDLE              = 00;
    parameter p_STATE_ANG1              = 01;
    parameter p_STATE_ANG2              = 02;
    parameter p_STATE_ANG3              = 03;
    parameter p_STATE_CASTSTART         = 04;
    parameter p_STATE_CASTWAIT          = 05;
    parameter p_STATE_ROMDEP1           = 06;
    parameter p_STATE_ROMDEP2           = 07;
    parameter p_STATE_ROMDEP3           = 08;
    parameter p_STATE_ROMDEP4           = 09;
    parameter p_STATE_ROMDEP5           = 10;
    parameter p_STATE_PIXELREND         = 11;
    parameter p_STATE_PIXELWAIT         = 12;

    // Inputs

    input                               clk;
    input                               reset;
    input                               i_Start;
    input                               i_IsWall;

    // Outputs

    output                              o_ForcePixLoad;
    output                              o_CmpPixLoad;
    output                              o_TryAICPixel;
    output                              o_UseAICTex;
    output                              o_LoadValid;
    output                              o_LoadHold;
    output                              o_LoadCoord;
    output                              o_WriteToMem;
    output                              o_ROMLoad;
    output                              o_Busy;
    output                              o_LoadCaster;
    output                              o_WallDelayLoad;
    output                              o_CasterIncrement;

    output [p_WIDTH_AICID:0]            o_AICID;
    output [p_WIDTH_SCREENY:0]          o_ScreenY;
    output [p_WIDTH_SELECT:0]           o_ROMSelect;
```

```
// Wires

wire                                w_PixelBusy;
wire                                w_PixelROMLoad;

// Internal Registers

reg                                 r_PixStart;
reg [3:0]                           r_State;

// Output Registers

reg                                 o_ROMLoad;
reg                                 o_LoadCaster;
reg                                 o_Busy;
reg                                 o_WallDelayLoad;
reg                                 o_CasterIncrement;
reg [p_WIDTH_SELECT:0]              o_ROMSelect;

// Instantiations

m_PixelRenderFSM pixrend(.clk(clk),
                    .reset(reset),
                    .i_Start(r_PixStart),
                    .o_ForcePixLoad(o_ForcePixLoad),
                    .o_ComparePixLoad(o_CmpPixLoad),
                    .o_Busy(w_PixelBusy),
                    .o_AICID(o_AICID),
                    .o_TryAICPixel(o_TryAICPixel),
                    .o_UseAICTex(o_UseAICTex),
                    .o_ROMLoad(w_PixelROMLoad),
                    .o_LoadValid(o_LoadValid),
                    .o_LoadHold(o_LoadHold),
                    .o_LoadCoord(o_LoadCoord),
                    .o_WriteVal(o_WriteToMem),
                    .o_ScreenY(o_ScreenY));

// Synchronous Block

always @ (posedge clk)
begin

    case (r_State)
        p_STATE_IDLE:       r_State = reset ? p_STATE_IDLE : i_Start ? p_STATE_ANG1 : p_STATE_IDLE;
        p_STATE_CASTWAIT:   r_State = reset ? p_STATE_IDLE : i_IsWall ? p_STATE_ROMDEP1 : p_STATE_CASTWAIT;
        p_STATE_PIXELWAIT:  r_State = reset ? p_STATE_IDLE : w_PixelBusy ? p_STATE_PIXELWAIT : p_STATE_IDLE;
        default:            r_State = reset ? p_STATE_IDLE : (r_State + 1);
    endcase

    case (r_State)
        p_STATE_ANG1:       o_ROMLoad = 1;
        p_STATE_CASTWAIT:   o_ROMLoad = 1;
        p_STATE_ROMDEP1:    o_ROMLoad = 1;
        p_STATE_PIXELWAIT:  o_ROMLoad = w_PixelROMLoad;
        default:            o_ROMLoad = 0;
    endcase

    case (r_State)
        p_STATE_ANG1:       o_ROMSelect = 1;
        p_STATE_CASTWAIT:   o_ROMSelect = 2;
        p_STATE_ROMDEP1:    o_ROMSelect = 5;
        p_STATE_PIXELWAIT:  o_ROMSelect = 3;
        default:            o_ROMSelect = 0;
    endcase

    r_PixStart          = (r_State == p_STATE_PIXELREND)    ? 1 : 0;
    o_Busy              = (r_State == p_STATE_IDLE)         ? 0 : 1;
    o_LoadCaster        = (r_State == p_STATE_CASTSTART)    ? 1 : 0;
    o_WallDelayLoad     = (r_State == p_STATE_CASTWAIT)     ? 1 : 0;
    o_CasterIncrement   = (r_State == p_STATE_CASTWAIT)     ? 1 : 0;
end
```

```
endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_HoldReg.v                                                                                  *
*   Description:    Holds values until they can be written to memory.                                            *
*   Date:           04/12/2004 11:30                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          None                                                                                         *
*                                                                                                                *
*****************************************************************************************************************/

module m_HoldReg(clk, i_InfoLoad, i_CoordLoad, i_ScreenX, i_ScreenY, i_Colour, i_Intensity, o_ScreenX, o_ScreenY,
                o_ColourID, o_Intensity);

    // Parameters

    parameter p_WIDTH_COLOURID          = (8  - 1);
    parameter p_WIDTH_INTENSITY         = (8  - 1);
    parameter p_WIDTH_SCREENX           = (9  - 1);
    parameter p_WIDTH_SCREENY           = (8  - 1);

    // Inputs

    input                               clk;

    input                               i_InfoLoad;
    input                               i_CoordLoad;

    input [p_WIDTH_SCREENX:0]           i_ScreenX;
    input [p_WIDTH_SCREENY:0]           i_ScreenY;

    input [p_WIDTH_COLOURID:0]          i_Colour;
    input [p_WIDTH_INTENSITY:0]         i_Intensity;

    // Ouptputs

    output [p_WIDTH_SCREENX:0]          o_ScreenX;
    output [p_WIDTH_SCREENY:0]          o_ScreenY;
    output [p_WIDTH_COLOURID:0]         o_ColourID;
    output [p_WIDTH_INTENSITY:0]        o_Intensity;

    // Output Registers

    reg [p_WIDTH_SCREENX:0]             o_ScreenX;
    reg [p_WIDTH_SCREENY:0]             o_ScreenY;
    reg [p_WIDTH_COLOURID:0]            o_ColourID;
    reg [p_WIDTH_INTENSITY:0]           o_Intensity;

    // Synchronous Block

    always @ (posedge clk)
    begin

        o_ScreenX   <= i_CoordLoad ? i_ScreenX   : o_ScreenX;
        o_ScreenY   <= i_CoordLoad ? i_ScreenY   : o_ScreenY;

        o_ColourID  <= i_InfoLoad  ? i_Colour    : o_ColourID;
        o_Intensity <= i_InfoLoad  ? i_Intensity : o_Intensity;

    end

endmodule
/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_InitialValueCalculator.v                                                                   *
*   Description:    Calculates the first boundary positions for the non-aligned values as well as retrieving    *
*                   the X and Y increments.                                                                      *
*   Date:           11/11/2004 11:43                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          21/11/2004 16:18 - First successful compilation                                              *
*                   21/11/2004 12:38 - Simulation debugging completed                                            *
*                                                                                                              *
***************************************************************************************************************/

module m_InitialValueCalculator(i_XIncrement, i_YIncrement, i_PositionX, i_PositionY,
                                o_InitialPositionX, o_InitialPositionY, o_XValid, o_YValid);

    // Parameters

    parameter p_WIDTH_MAPCOORDINATE     = (12 - 1);
    parameter p_WIDTH_INCREMENT         = (12 - 1);
    parameter p_WIDTH_ANGLE             = (11 - 1);
    parameter p_WIDTH_BUSDATA           = (8  - 1);
    parameter p_WIDTH_ADDRESS           = (8  - 1);
    parameter p_WIDTH_BYTENUMBEREQ      = (2  - 1);
    parameter p_WIDTH_BLOCKCOORDINATE   = (6  - 1);
    parameter p_WIDTH_INTERMEDIATEDATA  = (18 - 1);

    // Inputs

    input [p_WIDTH_INCREMENT:0]         i_XIncrement;
    input [p_WIDTH_INCREMENT:0]         i_YIncrement;

    input [p_WIDTH_MAPCOORDINATE:0]     i_PositionX;
    input [p_WIDTH_MAPCOORDINATE:0]     i_PositionY;

    // Outputs

    output [p_WIDTH_MAPCOORDINATE:0]    o_InitialPositionX;
    output [p_WIDTH_MAPCOORDINATE:0]    o_InitialPositionY;

    output                              o_XValid;
    output                              o_YValid;

    // Bidirectionals - none

    // Wires

    wire [p_WIDTH_INTERMEDIATEDATA:0]   w_XMultiplierOut;
    wire [p_WIDTH_INTERMEDIATEDATA:0]   w_YMultiplierOut;

    wire                                w_ChooseX;
    wire                                w_ChooseY;

    wire [p_WIDTH_BLOCKCOORDINATE:0]    w_ModuloOffsetX;
    wire [p_WIDTH_BLOCKCOORDINATE:0]    w_ModuloOffsetY;

    // Internal Registers

    // Output Registers

    // Module Instantiations

    m_multiplier6x12 xmult(.a(w_ModuloOffsetY),
                           .b(i_XIncrement),
                           .o(w_XMultiplierOut));

    m_multiplier6x12 ymult(.a(w_ModuloOffsetX),
                           .b(i_YIncrement),
                           .o(w_YMultiplierOut));

    // Assignments

    assign w_ModuloOffsetX = 64 - i_PositionX[p_WIDTH_BLOCKCOORDINATE:0];
    assign w_ModuloOffsetY = 64 - i_PositionY[p_WIDTH_BLOCKCOORDINATE:0];

    // Detect which outputs will be valid

    assign o_XValid = (i_XIncrement == 12'b100000000000) ? 0 : 1;
    assign o_YValid = (i_YIncrement == 12'b100000000000) ? 0 : 1;

    // Calculate Mux selectors
    assign w_ChooseX = (i_PositionY[p_WIDTH_BLOCKCOORDINATE:0] == 6'b000000) ? 1 : 0;
    assign w_ChooseY = (i_PositionX[p_WIDTH_BLOCKCOORDINATE:0] == 6'b000000) ? 1 : 0;

    // Select the wires

    assign o_InitialPositionX = i_PositionX + (w_ChooseX ? i_XIncrement : w_XMultiplierOut[17:6]);
    assign o_InitialPositionY = i_PositionY + (w_ChooseY ? i_YIncrement : w_YMultiplierOut[17:6]);

endmodule

/*****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_OneBitDelay.v                                                                             *
*   Description:    Delays one bit indefinately.                                                                *
*   Date:           03/12/2004 20:35                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          None                                                                                        *
*                                                                                                               *
*****************************************************************************************************************/

module m_OneBitDelay(clk, i_Value, i_Load, o_Value);

    // Inputs

    input                               clk;
    input                               i_Value;
    input                               i_Load;

    // Outputs

    output                              o_Value;

    // Output Registers

    reg                                 o_Value;
    reg                                 r_Value;

    // Synchronous Block

    always @ (posedge clk)
    begin

        // Load value if its ok to do so

        r_Value <= i_Load ? i_Value : r_Value;
        o_Value <= r_Value;

    end

endmodule

/*****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_ParametricIncrementRegister.v                                                             *
*   Description:    Holds and increments a value by adding an input to the held value every time the            *
*                   increment signal is asserted.                                                               *
*   Date:           11/11/2004 11:20                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          11/11/2004 11:42 - First successful compile                                                 *
*                   21/11/2004 19:38 - Debug - i_IncrementValue wrong width                                     *
*                   21/11/2004 22:38 - Modify - Added o_Overflow support                                        *
*                   21/11/2004 23:55 - Debug - Fixed o_Overflow                                                 *
*                                                                                                               *
*****************************************************************************************************************/

module m_ParametricIncrementRegister(clk, i_InitialCoordinate, i_IncrementValue,
                            i_LoadInitialValue, i_Increment, o_FinalCoordinate);

    // Parameters

    parameter p_WIDTH_MAPCOORDINATE     = (12 - 1);
    // Inputs

    input [p_WIDTH_MAPCOORDINATE:0]     i_InitialCoordinate;
    input [p_WIDTH_MAPCOORDINATE:0]     i_IncrementValue;
    input                               i_LoadInitialValue;
    input                               i_Increment;

    input                               clk;

    // Outputs

    output [p_WIDTH_MAPCOORDINATE:0]    o_FinalCoordinate;

    // Bidirectionals - none

    // Wires - none

    // Internal Registers

    reg [p_WIDTH_MAPCOORDINATE:0]       r_NewValue;

    // Output Registers

    reg [p_WIDTH_MAPCOORDINATE:0]       o_FinalCoordinate;

    // Synchronous Block

    always @ (posedge clk)
    begin

        // Calculate next value

        r_NewValue = i_IncrementValue + o_FinalCoordinate;

        // Set the output status

        o_FinalCoordinate = i_LoadInitialValue ? i_InitialCoordinate : i_Increment ? r_NewValue : o_FinalCoordinate;

    end
endmodule

/****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_PixelRegister.v                                                                           *
*   Description:    Takes in and decides pixel values.                                                          *
*   Date:           03/12/2004 20:03                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          None                                                                                        *
*                                                                                                               *
****************************************************************************************************************/

module m_PixelRegister(clk, i_ColourID, i_Depth, i_Intensity, i_Valid, i_ForceLoad, i_CompareLoad, i_PixelY,
                    o_ColourID, o_Intensity);

    // Parameters

    parameter p_WIDTH_COLOURID          = (8  - 1);
    parameter p_WIDTH_DEPTH             = (12 - 1);
    parameter p_WIDTH_INTENSITY         = (8  - 1);
    parameter p_WIDTH_SCREENY           = (8  - 1);

    // Inputs

    input [p_WIDTH_COLOURID:0]          i_ColourID;
    input [p_WIDTH_DEPTH:0]             i_Depth;
    input [p_WIDTH_INTENSITY:0]         i_Intensity;
    input                               i_Valid;
    input [p_WIDTH_SCREENY:0]           i_PixelY;

    input                               i_ForceLoad;
    input                               i_CompareLoad;
    input                               clk;

    // Outputs

    output [p_WIDTH_COLOURID:0]         o_ColourID;
    output [p_WIDTH_INTENSITY:0]        o_Intensity;

    // Bidirectionals

    // Wires

    wire                                w_CurrentCloser;
    wire [p_WIDTH_INTENSITY:0]          w_DefIntensity;
    wire [p_WIDTH_COLOURID:0]           w_DefColID;

    // Internal Registers

    reg                                 r_ColourValid;
    reg                                 r_AllValid;
    reg                                 r_SpecialLoad;
    reg [p_WIDTH_DEPTH:0]               r_CurrentDepth;

    // Output Registers

    reg [p_WIDTH_COLOURID:0]            o_ColourID;
    reg [p_WIDTH_INTENSITY:0]           o_Intensity;

    // Instantiations

    m_cmpl12 comparedepth(.A(r_CurrentDepth), .B(i_Depth), .A_LT_B(w_CurrentCloser));

    m_floorsram16x240 wfram(.addr(i_PixelY), .dout({w_DefColID, w_DefIntensity}), .clk(clk));

    // Syncronous Block

    always @ (posedge clk)
    begin

        // Check the colour

        r_ColourValid   = (i_ColourID == 0) ? 0 : 1;

        // Check overall validity

        r_AllValid      = (i_CompareLoad & r_ColourValid & (~w_CurrentCloser) & i_Valid) | i_ForceLoad;
        r_SpecialLoad   = (~i_Valid) & i_ForceLoad;

        // Chnage stored values

        r_CurrentDepth  = r_AllValid ? i_Depth : r_CurrentDepth;
        o_ColourID      = r_SpecialLoad ? 8'h00 : r_AllValid ? i_ColourID : o_ColourID;
        //r_SpecialLoad ? 8'h01 : i_ForceLoad ? 8'h02 : o_ColourID;
        o_Intensity     = r_SpecialLoad ? 8'h40/*w_DefIntensity*/ : r_AllValid ? i_Intensity : o_Intensity;

    end

endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_PixelRenderFSM.v                                                                           *
*   Description:    Renders a single pixel.                                                                      *
*   Date:           03/12/2004 23:54                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          None                                                                                         *
*                                                                                                                *
*****************************************************************************************************************/

//   ----------------------|
//   |                     |
//   |                     v
//  Wall----->YtoV-------->ROM----->PixelReg---->HoldReg---->VideoRAM
//             ^       |    ^          ^
//             |       |    |          |
//             |   ----|-----          |
//     ---------   |   ------          |
//     |           |        |          |
//     |           |        v          |
//  AICRAM------>XtoU---->Delay--------|

//                  I W
//                  D A
//                  L I  S S S S S S S S  R R R R R R R R  E E E E E E E E  F F F
//                  E T  0 1 2 3 4 5 6 7  0 1 2 3 4 5 6 7  0 1 2 3 4 5 6 7  0 1 2
//
// o_ForcePixLoad   X X  0 1 0 0 0 0 0 0  0 1 0 0 0 0 0 0  0 1 0 0 0 0 0 0  0 0 0
// o_ComparePixLoad X X  0 0 1 1 1 1 1 1  1 0 1 1 1 1 1 1  1 0 1 1 1 1 1 1  1 0 0
// o_Busy           0 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1  1 1 0
// o_AICID          X 1  2 3 4 5 6 7 X 1  2 3 4 5 6 7 X 1  2 3 4 5 6 7 X X  X X X
// o_TryAICPixel    0 0  0 1 1 1 1 1 1 1  0 1 1 1 1 1 1 1  0 1 1 1 1 1 1 1  X X X
// o_UseAICTex      1 0  1 1 1 1 1 1 1 0  1 1 1 1 1 1 1 0  1 1 1 1 1 1 X X  X X
// o_ROMLoad        0 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 0  0 0
// o_LoadValid      0 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 0  0 0
// o_LoadHold       0 0  0 0 0 0 0 0 0 0  0 1 0 0 0 0 0 0  0 1 0 0 0 0 0 0  0 1 0
// o_LoadCoord      0 0  0 0 0 0 0 0 1 0  0 0 0 0 0 0 1 0  0 0 0 0 0 0 1 0  0 0
// o_WriteVal       0 0  0 0 0 0 0 0 0 0  0 0 1 0 0 0 0 0  0 0 1 0 0 0 0 0  0 0 1
// o_ScreenY        0 K  K K K K K K + K  K K K K K K + K  K K K K K K + K  0 0

module m_PixelRenderFSM(clk, reset, i_Start, o_ForcePixLoad, o_ComparePixLoad, o_Busy, o_AICID, o_TryAICPixel,
                o_UseAICTex, o_ROMLoad, o_LoadValid, o_LoadHold, o_LoadCoord, o_WriteVal, o_ScreenY);

    // Parameters

    parameter p_WIDTH_AICID             = (3  - 1);
    parameter p_WIDTH_SCREENY           = (8  - 1);

    parameter p_STATE_IDLE              = 00;
    parameter p_STATE_WAIT              = 01;
    parameter p_STATE_S0                = 02;
    parameter p_STATE_S1                = 03;
    parameter p_STATE_S2                = 04;
    parameter p_STATE_S3                = 05;
    parameter p_STATE_S4                = 06;
    parameter p_STATE_S5                = 07;
    parameter p_STATE_S6                = 08;
    parameter p_STATE_S7                = 09;
    parameter p_STATE_R0                = 10;
    parameter p_STATE_R1                = 11;
    parameter p_STATE_R2                = 12;
    parameter p_STATE_R3                = 13;
    parameter p_STATE_R4                = 14;
    parameter p_STATE_R5                = 15;
    parameter p_STATE_R6                = 16;
    parameter p_STATE_R7                = 17;
    parameter p_STATE_E0                = 18;
    parameter p_STATE_E1                = 19;
    parameter p_STATE_E2                = 20;
    parameter p_STATE_E3                = 21;
    parameter p_STATE_E4                = 22;
    parameter p_STATE_E5                = 23;
    parameter p_STATE_E6                = 24;
    parameter p_STATE_E7                = 25;
    parameter p_STATE_F0                = 26;
    parameter p_STATE_F1                = 27;
    parameter p_STATE_F2                = 28;

    // Inputs

    input                               i_Start;

    input                               clk;
    input                               reset;

    // Outputs
```

```
output                              o_ForcePixLoad;
output                              o_ComparePixLoad;
output                              o_Busy;
output [p_WIDTH_AICID:0]            o_AICID;
output                              o_TryAICPixel;
output                              o_UseAICTex;
output                              o_ROMLoad;
output                              o_LoadValid;
output                              o_LoadHold;
output                              o_LoadCoord;
output                              o_WriteVal;
output [p_WIDTH_SCREENY:0]          o_ScreenY;

// Internal Registers

reg [4:0]                           r_State;

// Output Registers

reg                                 o_ForcePixLoad;
reg                                 o_Busy;
reg [p_WIDTH_AICID:0]               o_AICID;
reg                                 o_TryAICPixel;
reg                                 o_UseAICTex;
reg                                 o_ROMLoad;
reg                                 o_LoadHold;
reg                                 o_LoadCoord;
reg                                 o_WriteVal;
reg [p_WIDTH_SCREENY:0]             o_ScreenY;

// Synchronous Block

always @ (posedge clk)
begin

    // Set next state

    case (r_State)
        p_STATE_IDLE:   r_State = reset ? p_STATE_IDLE : i_Start ? p_STATE_WAIT : p_STATE_IDLE;
        p_STATE_R7:     r_State = reset ? p_STATE_IDLE : (o_ScreenY == 239) ? p_STATE_E0 : p_STATE_R0;
        p_STATE_F2:     r_State = p_STATE_IDLE;
        default:        r_State = reset ? p_STATE_IDLE : (r_State + 1);
    endcase

    case (r_State)
        p_STATE_S1:     o_ForcePixLoad = 1;
        p_STATE_R1:     o_ForcePixLoad = 1;
        p_STATE_E1:     o_ForcePixLoad = 1;
        default:        o_ForcePixLoad = 0;
    endcase

    case (r_State)
        p_STATE_IDLE:   o_Busy = 0;
        p_STATE_F2:     o_Busy = 0;
        default:        o_Busy = 1;
    endcase

    o_AICID = (r_State == p_STATE_IDLE) ? 0 : (o_AICID + 1);

    case (r_State)
        p_STATE_WAIT:   o_TryAICPixel = 0;
        p_STATE_S7:     o_TryAICPixel = 0;
        p_STATE_R7:     o_TryAICPixel = 0;
        default:        o_TryAICPixel = 1;
    endcase

    case (r_State)
        p_STATE_WAIT:   o_UseAICTex = 0;
        p_STATE_S7:     o_UseAICTex = 0;
        p_STATE_R7:     o_UseAICTex = 0;
        default:        o_UseAICTex = 1;
        endcase

        case (r_State)
            p_STATE_IDLE:   o_ROMLoad = 0;
            p_STATE_E7:     o_ROMLoad = 0;
            p_STATE_F0:     o_ROMLoad = 0;
            p_STATE_F1:     o_ROMLoad = 0;
            default:        o_ROMLoad = 1;
        endcase

        case (r_State)
            p_STATE_R1:     o_LoadHold = 1;
            p_STATE_E1:     o_LoadHold = 1;
            p_STATE_F1:     o_LoadHold = 1;
            default:        o_LoadHold = 0;
        endcase

        case (r_State)
            p_STATE_S6:     o_LoadCoord = 1;
            p_STATE_R6:     o_LoadCoord = 1;
            p_STATE_E6:     o_LoadCoord = 1;
            default:        o_LoadCoord = 0;
        endcase

        case (r_State)
            p_STATE_R2:     o_WriteVal = 1;
            p_STATE_E2:     o_WriteVal = 1;
            p_STATE_F2:     o_WriteVal = 1;
            default:        o_WriteVal = 0;
        endcase

        case (r_State)
            p_STATE_IDLE:   o_ScreenY = 0;
            p_STATE_S6:     o_ScreenY = (o_ScreenY + 1);
            p_STATE_R6:     o_ScreenY = (o_ScreenY + 1);
            default:        o_ScreenY = o_ScreenY;
        endcase
    end

    assign o_ComparePixLoad     = ~o_ForcePixLoad;
    assign o_LoadValid          = o_ROMLoad;

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_PositionCalcFsm.v                                                             *
*  Description:     Controls the new AI and Player positions                                        *
*  Date:            03/12/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           03/12/04 20.39 First Draft                                                      *
*                                                                                                   *
*                                                                                                   *
****************************************************************************************************/

module m_PositionCalcFsm(clk, reset, i_Character, i_PosCalcStart, i_PCurrentStatus, i_AICurrentStatus,
                         i_PData, i_AIData, i_PDirection, i_IsWall, i_AIValidData, o_PosCalcBusy, o_PNewStatus,
                         o_AINewStatus, o_Character, o_FetchWallInfo, o_WallX, o_WallY, o_IsSide, o_PDataClr,
                         o_AIDataClr, o_DirectionLookup, o_LEPlayerPosition, o_WEAICPosition, o_AIReq, o_ptry);

   // state parameters
   parameter              p_STATEIDLE             = 0;
   parameter              p_STATECURRENTPOSITION  = 1;
   parameter              p_STATENEWPOSITION      = 2;
   parameter              p_STATECALCPOSITION     = 3;
   parameter              p_STATEWALLCHECK        = 4;
   parameter              p_STATEPOSOUT           = 5;

   // module parameters
   parameter              p_LEFT                  = 1;
   parameter              p_RIGHT                 = 2;
   parameter              p_UP                    = 3;
   parameter               p_DOWN                  = 4;
   parameter               p_TURNLEFT              = 5;
   parameter               p_TURNRIGHT             = 6;

   parameter               p_UPLEFT                = 5;
   parameter               p_UPRIGHT               = 6;
   parameter               p_DOWNLEFT              = 7;
   parameter               p_DOWNRIGHT             = 8;

   parameter               p_ANGLEINCREMENT        = 16;
   parameter               p_AIINCREMENTSTRAIGHT   = 2;
   parameter               p_AIINCREMENT           = 1;

   // module inputs
   input                   clk;
   input                   reset;
   input                   i_PosCalcStart;
   input [34:0]            i_PCurrentStatus;
   input [23:0]            i_AICurrentStatus;
   input [2:0]             i_PData;             // New position request from the keyboard
   input [3:0]             i_AIData;            // New AI position from the AI Block (left, right, up etc...)
   input [9:0]             i_PDirection;
   input                   i_IsWall;
   input [2:0]             i_Character;
   input                   i_AIValidData;

   // module outputs
   output                  o_PosCalcBusy;
   output [34:0]           o_PNewStatus;
   output [23:0]           o_AINewStatus;
   output [2:0]            o_Character;
   output                  o_FetchWallInfo;
   output [5:0]            o_WallX;
   output [5:0]            o_WallY;
   output                  o_IsSide;
   output                  o_PDataClr;
   output                  o_AIDataClr;
   output [6:0]            o_DirectionLookup;
   output                  o_LEPlayerPosition;
   output                  o_WEAICPosition;
   output                  o_AIReq;

   output [11:0]           o_ptry;
   reg [11:0]              o_ptry;

   // internal registers
   reg [2:0]               r_State;
   reg [3:0]               r_PosReq;
   reg [9:0]               r_PosChange;
   reg [11:0]              r_PCurrentX;
   reg [11:0]              r_PCurrentY;
   reg [10:0]              r_PCurrentT;
   reg [11:0]              r_PNewX;
   reg [11:0]              r_PNewY;
   reg [10:0]              r_PNewT;
   reg [11:0]              r_AICurrentX;
   reg [11:0]              r_AICurrentY;
   reg [11:0]              r_AINewX;
   reg [11:0]              r_AINewY;

   // output registers
   reg                     o_PosCalcBusy;
   reg [34:0]              o_PNewStatus;
   reg [23:0]              o_AINewStatus;
   reg [2:0]               o_Character;
   reg                     o_FetchWallInfo;
   reg [5:0]               o_WallX;
   reg [5:0]               o_WallY;
   reg                     o_IsSide;
   reg                     o_PDataClr;
   reg                     o_AIDataClr;
   reg [6:0]               o_DirectionLookup;
   reg                     o_LEPlayerPosition;
   reg                     o_WEAICPosition;
   reg                     o_AIReq;


   always @ (posedge clk or posedge reset)
   begin
      if (reset)
      begin
         r_State  <= p_STATEIDLE;
      end // if statement
      else
      begin
         case (r_State)

         p_STATEIDLE:
         begin
            o_PosCalcBusy     <= 0;
            o_PNewStatus      <= 0;
            o_AINewStatus     <= 0;
            o_Character       <= 0;
            o_FetchWallInfo   <= 0;
            o_WallX           <= 0;
            o_WallY           <= 0;
            o_IsSide          <= 0;
            if (i_PosCalcStart == 1)
            begin
               r_State  <= p_STATECURRENTPOSITION;
               o_PosCalcBusy <= 1;
            end
            else
               r_State  <= p_STATEIDLE;
            o_LEPlayerPosition   <= 0;
            o_WEAICPosition   <= 0;
         end // p_STATEIDLE case

         p_STATECURRENTPOSITION:
         begin
            o_PosCalcBusy <= 1;
            if (i_Character == 0) // Player update
            begin
               r_PosReq          <= {1'b0,i_PData};                  // read request from keyboard
               r_PCurrentX       <= i_PCurrentStatus[34:23];
               r_PCurrentY       <= i_PCurrentStatus[22:11];
               r_PCurrentT       <= i_PCurrentStatus[10:0];    // read current position of player
               r_State           <= p_STATENEWPOSITION;
            end // if statement
            else                 // Character update
               o_AIReq           <= 1;                         // read request for direction from AI block
               o_Character       <= i_Character;
               r_AICurrentX      <= i_AICurrentStatus[23:12];
               r_AICurrentY      <= i_AICurrentStatus[11:0];   // read current position of AIC
               r_State           <= p_STATENEWPOSITION;
            begin
            end // else statement
         end // p_STATECURRENTPOSITION case

         p_STATENEWPOSITION:
         begin
            o_PosCalcBusy <= 1;
            if (i_Character == 0)
            begin
               o_PDataClr           <= 1;                    // clear PDataReg since request dealt with
               o_DirectionLookup     = r_PCurrentT[10:4];   // lookup standard jumps for given player orientation
               r_PosChange          <= i_PDirection;        // set change in position from lookup
               r_State              <= p_STATECALCPOSITION;
            end // if statement
            else
            begin
               if (i_AIValidData == 0)
               begin
                  r_State  <= p_STATECALCPOSITION;
               end
               else
               begin
                  r_PosReq    <= i_AIData;
                  o_AIDataClr <= 1;                   // clear AIDataReg since request dealt with
                  r_State     <= p_STATECALCPOSITION;
               end // else statement
            end // else statement
            o_LEPlayerPosition   <= 0;
            o_WEAICPosition   <= 0;
         end // p_STATENEWPOSITION case

         p_STATECALCPOSITION:
         begin
            o_PosCalcBusy <= 1;
            o_ptry = {7'h00, r_PosChange[9:5]};
            if (i_Character == 0)
            begin
               case (r_PosReq)
                  p_LEFT:
                  begin
                     r_PNewX = r_PCurrentX - (r_PosChange[4:0]);
                     r_PNewY = r_PCurrentY - (r_PosChange[9:5]);
                     r_PNewT = r_PCurrentT;
                  end

                  p_RIGHT:
                  begin
                     r_PNewX = r_PCurrentX + (r_PosChange[4:0]);
                     r_PNewY = r_PCurrentY + (r_PosChange[9:5]);
                     r_PNewT = r_PCurrentT;
                  end

                  p_UP:
                  begin
                     r_PNewX = r_PCurrentX + (r_PosChange[9:5]);
                     r_PNewY = r_PCurrentY - (r_PosChange[4:0]);
                     r_PNewT = r_PCurrentT;
                  end

                  p_DOWN:
                  begin
                     r_PNewX = r_PCurrentX - (r_PosChange[9:5]);
                     r_PNewY = r_PCurrentY + (r_PosChange[4:0]);
                     r_PNewT = r_PCurrentT;
                  end

                  p_TURNLEFT:
                  begin
                     r_PNewX   = r_PCurrentX;
                     r_PNewY   = r_PCurrentY;
                     r_PNewT   = r_PCurrentT - p_ANGLEINCREMENT;
                  end

                  p_TURNRIGHT:
                  begin
                     r_PNewX   = r_PCurrentX;
                     r_PNewY   = r_PCurrentY;
                     r_PNewT   = r_PCurrentT + p_ANGLEINCREMENT;
                  end

                  default:
                  begin
                     r_PNewX  = r_PCurrentX;
                     r_PNewY  = r_PCurrentY;
                     r_PNewT  = r_PCurrentT;
                     r_State  <= p_STATEIDLE;
                  end
               endcase

               if (r_PNewX[5] == 1)
               begin
                  o_WallX           <= r_PNewX[11:6] + 1;
                  o_WallY           <= r_PNewY[11:6];
                  o_IsSide          <= 1;                   // test for side wall
                  o_FetchWallInfo   <= 1;
               end
               else
               begin
                  o_WallX           <= r_PNewX[11:6];
                  o_WallY           <= r_PNewY[11:6];
                  o_IsSide          <= 1;                   // test for side wall
                  o_FetchWallInfo   <= 1;
               end

               r_State <= p_STATEWALLCHECK;
            end // if statement
            else
            begin
               case (r_PosReq)
               p_LEFT:
               begin
                  r_AINewX = r_AICurrentX - p_AIINCREMENTSTRAIGHT;
                  r_AINewY = r_AICurrentY;
               end

               p_RIGHT:
               begin
                  r_AINewX = r_AICurrentX + p_AIINCREMENTSTRAIGHT;
                  r_AINewY = r_AICurrentY;
               end

               p_UP:
               begin
                  r_AINewX = r_AICurrentX;
                  r_AINewY = r_AICurrentY + p_AIINCREMENTSTRAIGHT;
               end

               p_DOWN:
               begin
                  r_AINewX = r_AICurrentX;
                  r_AINewY = r_AICurrentY - p_AIINCREMENTSTRAIGHT;
               end

               p_UPLEFT:
               begin
                  r_AINewX = r_AICurrentX - p_AIINCREMENT;
                  r_AINewY = r_AICurrentY - p_AIINCREMENT;
               end

               p_UPRIGHT:
               begin
                  r_AINewX = r_AICurrentX + p_AIINCREMENT;
                  r_AINewY = r_AICurrentY - p_AIINCREMENT;
               end

               p_DOWNLEFT:
               begin
                  r_AINewX = r_AICurrentX - p_AIINCREMENT;
                  r_AINewY = r_AICurrentY + p_AIINCREMENT;
               end

               p_DOWNRIGHT:
               begin
                  r_AINewX = r_AICurrentX + p_AIINCREMENT;
                  r_AINewY = r_AICurrentY + p_AIINCREMENT;
               end

               default:
               begin
                  r_AINewX = r_AICurrentX;
                  r_AINewY = r_AICurrentY;
               end
               endcase
```

```
      if (r_AINewX[5] == 1)
      begin
         o_WallX           <= r_AINewX[11:6] + 1;
         o_WallY           <= r_AINewY[11:6] ;
         o_IsSide          <= 1;                   // test for side wall
         o_FetchWallInfo   <= 1;
      end // else statement
      else
      begin
         o_WallX           <= r_AINewX[11:6];
         o_WallY           <= r_AINewY[11:6];
         o_IsSide          <= 1;                   // test for side wall
         o_FetchWallInfo   <= 1;
      end
      r_State <= p_STATEWALLCHECK;
   end // else statement
   o_LEPlayerPosition   <= 0;
   o_WEAICPosition   <= 0;
end // p_STATECALCPOSITION case

p_STATEWALLCHECK:       // receive results from side wall check, now check appropriate top wall
                        //   and output current positions
begin
   o_PosCalcBusy <= 1;
   if (i_Character == 0)
   begin
      if (i_IsWall == 1)
      begin
         r_PNewX = r_PCurrentX;
      end
      else
      begin
         r_PNewX = r_PNewX;
      end

      if (r_PNewY[5] == 1)
      begin
         o_WallX           <= r_PNewX[11:6];
         o_WallY           <= r_PNewY[11:6] + 1;
         o_IsSide          <= 0;                   // test for top wall
         o_FetchWallInfo   <= 1;
      end // if statement
      else
      begin
         o_WallX           <= r_PNewX[11:6];
         o_WallY           <= r_PNewY[11:6];
         o_IsSide          <= 0;                   // test for top wall
         o_FetchWallInfo   <= 1;
      end // else statement
   end
   else
   begin
      if (i_IsWall == 1)
      begin
         r_AINewX = r_AICurrentX;
      end
      else
      begin
         r_AINewX = r_AINewX;
      end

      if (r_AINewY[5] == 1)
      begin
         o_WallX           <= r_PNewX[11:6];
         o_WallY           <= r_PNewY[11:6] + 1;
         o_IsSide          <= 0;                   // test for top wall
         o_FetchWallInfo   <= 1;
      end // if statement
      else
      begin
         o_WallX           <= r_PNewX[11:6];
```

```
                o_WallY           <= r_PNewY[11:6];
                o_IsSide          <= 0;                   // test for top wall
                o_FetchWallInfo   <= 1;
             end // else statement
          end
          o_LEPlayerPosition   <= 0;
          o_WEAICPosition   <= 0;
          r_State <= p_STATEPOSOUT;
       end // p_STATEWALLCHECK case

       p_STATEPOSOUT:
       begin
          o_PosCalcBusy  <= 1;
          if (i_Character == 0)
          begin
             if (i_IsWall == 1)
             begin
                r_PNewY = r_PCurrentY;
             end
             else
             begin
                r_PNewY = r_PNewY;
             end
             o_LEPlayerPosition   <= 1;
             o_WEAICPosition   <= 0;
             o_PNewStatus         <= {r_PNewX, r_PNewY, r_PNewT};
          end // if statement
          else
          begin
             if (i_IsWall == 1)
             begin
                r_AINewY = r_AICurrentY;
             end
             else
             begin
                r_AINewY = r_AINewY;
             end
             o_LEPlayerPosition   <= 0;
             o_WEAICPosition   <= 1;
             o_Character       <= i_Character;
             o_AINewStatus     <= {r_AINewX, r_AINewY};
          end
          r_State     <= p_STATEIDLE;
       end // p_STATEPOSOUT

       default: r_State  <= p_STATEIDLE;
       endcase
    end // else statement
  end // else statement
endmodule

/**************************************************************************************************************
*                                                                                                             *
*   Filename:       m_PositionToDepthSquared.v                                                                *
*   Description:    Converts character and player positions to a distance squared.                           *
*   Date:           23/11/2004 14:57                                                                          *
*   Author:         Robert Cassels                                                                            *
*   Notes:          23/11/2004 16:21 - First compiled successfully                                            *
*                                                                                                             *
**************************************************************************************************************/

module m_PositionToDepthSquared(i_PlayerX, i_PlayerY, i_CharacterX, i_CharacterY,
                                o_DifferenceX, o_DifferenceY, o_DepthSquared);

   // Parameters

   parameter p_WIDTH_POSITION          = (12 - 1);
   parameter p_WIDTH_DEPTHSQUARE       = (12 - 1);
   parameter p_WIDTH_DIFFERENCE        = (13 - 1);
   parameter p_WIDTH_FULLSQUARE        = (26 - 1);

   // Inputs
```

```
    input [p_WIDTH_POSITION:0]          i_PlayerX;
    input [p_WIDTH_POSITION:0]          i_PlayerY;
    input [p_WIDTH_POSITION:0]          i_CharacterX;
    input [p_WIDTH_POSITION:0]          i_CharacterY;

    // Outputs

    output [p_WIDTH_DIFFERENCE:0]       o_DifferenceX;
    output [p_WIDTH_DIFFERENCE:0]       o_DifferenceY;
    output [p_WIDTH_DEPTHSQUARE:0]      o_DepthSquared;

    // Bidirectionals

    // Wires

    wire [p_WIDTH_FULLSQUARE:0]         w_SquareX;
    wire [p_WIDTH_FULLSQUARE:0]         w_SquareY;
    wire [p_WIDTH_FULLSQUARE:0]         w_SquareSum;

    // Set the differences

    assign o_DifferenceX = {1'b0, i_PlayerX} - {1'b0, i_CharacterX};
    assign o_DifferenceY = {1'b0, i_PlayerY} - {1'b0, i_CharacterY};

    // Instantiation

    m_multiplier13x13 multx(.o(w_SquareX), .a(o_DifferenceX), .b(o_DifferenceX));
    m_multiplier13x13 multy(.o(w_SquareY), .a(o_DifferenceY), .b(o_DifferenceY));

    // Assign the square sum

    assign w_SquareSum = w_SquareX + w_SquareY;

    // Now Create the correctly encoded value

    assign o_DepthSquared = (w_SquareSum[25]    != 0) ? {12'b111111111111} :
                            (w_SquareSum[24:23] != 0) ? {w_SquareSum[24:16], 3'b111} :
                            (w_SquareSum[22:21] != 0) ? {w_SquareSum[22:14], 3'b110} :
                            (w_SquareSum[20:19] != 0) ? {w_SquareSum[20:12], 3'b101} :
                            (w_SquareSum[18:17] != 0) ? {w_SquareSum[18:10], 3'b100} :
                            (w_SquareSum[16:15] != 0) ? {w_SquareSum[16:08], 3'b011} :
                            (w_SquareSum[14:13] != 0) ? {w_SquareSum[14:06], 3'b010} :
                            (w_SquareSum[12:11] != 0) ? {w_SquareSum[12:04], 3'b001} :
                                                        {w_SquareSum[10:02], 3'b000} ;

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_PositionUpdate.v                                                              *
*  Description:     Controls the new AI and Player positions and attacks                            *
*  Date:            03/12/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           03/12/04 17.50 First Draft                                                      *
*                                                                                                   *
*                                                                                                   *
****************************************************************************************************/


module m_PositionUpdate(clk, reset, i_PPositionStatus, i_AIPositionStatus, i_PValidData, i_ControlCode,
                        i_AIValidData, i_AIDirection, i_IsWall, i_PosUpdateStart, i_AttackBusy,
                        o_PNewStatus, o_LEPlayerPosition, o_AINewStatus, o_WEAICPosition, o_Character,
                        o_FetchWallInfo, o_WallX, o_WallY, o_IsSide, o_AttackStart, o_PosUpdateBusy,
                         w_PosCalcBusy, o_ptry);

   output w_PosCalcBusy;
   output [11:0] o_ptry;

   // parameters - none

   // module inputs
```

```
input                   clk;
input                   reset;
input [34:0]            i_PPositionStatus;
input [23:0]            i_AIPositionStatus;
input                   i_PValidData;
input [2:0]             i_ControlCode;
input                   i_AIValidData;
input [2:0]             i_AIDirection;
input                   i_IsWall;
input                   i_PosUpdateStart;
input                   i_AttackBusy;

// module outputs
output [34:0]           o_PNewStatus;     // to Player Position
output                  o_LEPlayerPosition;
output [23:0]           o_AINewStatus;    // to AI Position RAM
output                  o_WEAICPosition;
output [2:0]            o_Character;      // index for AI Position RAM
output                  o_FetchWallInfo;  // to 3D Renderer
output [5:0]            o_WallX;
output [5:0]            o_WallY;
output                  o_IsSide;
output                  o_AttackStart;    // to Attack Control
output                  o_PosUpdateBusy;

// module wires
wire [2:0]              w_Character;
wire                    w_PosCalcStart;
wire [2:0]              w_PData;
wire [2:0]              w_AIData;
wire [9:0]              w_PDirection;
wire                    w_PosCalcBusy;
wire                    w_PDataClr;
wire                    w_AIDataClr;
wire [6:0]              w_DirectionLookup;



m_PositionCalcFsm s_PositionCalcFsm(.clk(clk),
                     .reset(reset),
                     .i_Character (w_Character),
                     .i_PosCalcStart (w_PosCalcStart),
                     .i_PCurrentStatus(i_PPositionStatus),
                     .i_AICurrentStatus(i_AIPositionStatus),
                     .i_PData(w_PData),
                     .i_AIValidData(i_AIValidData),
                     .i_AIData(w_AIData),
                     .i_PDirection(w_PDirection),
                     .i_IsWall(i_IsWall),
                     .o_PosCalcBusy(w_PosCalcBusy),
                     .o_PNewStatus(o_PNewStatus),
                     .o_AINewStatus(o_AINewStatus),
                     .o_Character(o_Character),
                     .o_FetchWallInfo(o_FetchWallInfo),
                     .o_WallX(o_WallX),
                     .o_WallY(o_WallY),
                     .o_IsSide(o_IsSide),
                     .o_PDataClr(w_PDataClr),
                     .o_AIDataClr(w_AIDataClr),
                     .o_DirectionLookup(w_DirectionLookup),
                     .o_LEPlayerPosition(o_LEPlayerPosition),
                     .o_WEAICPosition(o_WEAICPosition)
                     );

m_PositionUpdateFsm     s_PositionUpdateFsm(.clk(clk),
                                         .reset(reset),
                                         .i_PosUpdateStart(i_PosUpdateStart),
                                         .i_PosCalcBusy(w_PosCalcBusy),
                                         .i_AttackBusy(i_AttackBusy),
                                         .o_Character(w_Character),
                                         .o_PosCalcStart(w_PosCalcStart),
                                         .o_AttackStart(o_AttackStart),
                                                .o_PosUpdateBusy(o_PosUpdateBusy));

   assign o_ptry = {6'h00, w_PDirection};

   m_pdirection            s_pdirection(.addr(i_PPositionStatus[10:4]), .clk(clk), .dout(w_PDirection));
   m_PURegister3           s_PDataReg(clk, i_PValidData, 1'b0, i_ControlCode, w_PData); //w_PDataClr
   m_PURegister3           s_AIDataReg(clk, i_AIValidData, w_AIDataClr, i_AIDirection, w_AIData);

endmodule

/****************************************************************************************************
*                                                                                                   *
*  Filename:        m_PositionUpdateFsm.v                                                           *
*  Description:     Controls the new AI and Player positions and attacks                            *
*  Date:            03/12/04                                                                        *
*  Author:          Tim Mamtora                                                                     *
*  Notes:           03/12/04 17.50 First Draft                                                      *
*                                                                                                   *
*                                                                                                   *
****************************************************************************************************/


module m_PositionUpdateFsm(clk, reset, i_PosUpdateStart, i_PosCalcBusy, i_AttackBusy, o_Character, o_PosCalcStart, o_AttackStar

   // state parameters
   parameter           p_STATEIDLE                 = 0;
   parameter           p_STATEPOSCALCSTART         = 1;
   parameter           p_STATEBUSYWAIT                = 6;
   parameter           p_STATEPOSCALCWAIT          = 2;
   parameter           p_STATEWAIT                 = 3;
   parameter           p_STATEATTACKSTART          = 4;
   parameter           p_STATEATTACKWAIT           = 5;

   // module inputs
   input               clk;
   input               reset;
   input               i_PosUpdateStart;
   input               i_PosCalcBusy;
   input               i_AttackBusy;

   // module outputs
   output              o_PosCalcStart;
   output              o_AttackStart;
   output              o_PosUpdateBusy;
   output [2:0]        o_Character;

   // internal registers
   reg [2:0]           r_State;
   reg [2:0]           r_Character;

   // output registers
   reg              o_PosCalcStart;
   reg              o_AttackStart;
   reg              o_PosUpdateBusy;
   reg   [2:0]      o_Character;

   always @ (posedge clk or posedge reset)
   begin
      if (reset)
      begin
         o_PosCalcStart    <= 0;
         o_AttackStart     <= 0;
         o_PosUpdateBusy   <= 0;
         o_Character       <= 0;
         r_State           <= p_STATEIDLE;
         r_Character       <= 0;
      end // if statement
      else
      begin
         case (r_State)

         p_STATEIDLE:
```

```
begin
   o_PosCalcStart    <= 0;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 0;
   o_Character       <= 0;
   r_Character       <= 0;
   if (i_PosUpdateStart == 1)
      r_State <= p_STATEPOSCALCSTART;
   else
      r_State <= p_STATEIDLE;
end // p_STATEIDLE case

p_STATEPOSCALCSTART:
begin
   o_PosCalcStart    <= 1;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 1;
   r_State           <= p_STATEBUSYWAIT;
end // p_STATEPOSCALCSTART case

p_STATEBUSYWAIT:
begin
   o_PosCalcStart    <= 1;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 1;
   r_State           <= p_STATEPOSCALCWAIT;
end

p_STATEPOSCALCWAIT:
begin
   o_PosCalcStart    <= 0;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 1;
   if (i_PosCalcBusy == 1)
      r_State  <= p_STATEPOSCALCWAIT;
   else
      r_State  <= p_STATEWAIT;
end // p_STATEPOSCALCWAIT

p_STATEWAIT:
begin
   o_PosCalcStart    <= 0;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 1;
   r_State           <= p_STATEATTACKSTART;
end // p_STATEWAIT

p_STATEATTACKSTART:
begin
   o_PosCalcStart    <= 0;
   o_AttackStart     <= 1;
   o_PosUpdateBusy   <= 1;
   o_Character       <= r_Character;
   r_State           <= p_STATEATTACKWAIT;
end // p_STATEATTACKSTART

p_STATEATTACKWAIT:
begin
   o_PosCalcStart    <= 0;
   o_AttackStart     <= 0;
   o_PosUpdateBusy   <= 1;
   if (i_AttackBusy == 1)
      r_State  <= p_STATEATTACKWAIT;
   else
      if(r_Character == 7)
         r_State  <= p_STATEIDLE;
      else
      begin
         r_Character <= r_Character + 1;
         r_State     <= p_STATEPOSCALCSTART;
      end // else statement
end // p_STATEATTACKWAIT case
```

```
         default: r_State  <= p_STATEIDLE;
         endcase
      end // else statement
   end // always block
endmodule

/*****************************************************************************************************
*                                                                                                    *
*  Filename:         m_PURegister3.v                                                                 *
*  Description:      3-bit Register with clear                                                       *
*  Date:             03/12/04                                                                        *
*  Author:           Tim Mamtora                                                                     *
*  Notes:            03/12/04 20:28 First Draft                                                      *
*                                                                                                    *
*****************************************************************************************************/

module m_PURegister3(clk, i_ld, i_clr, i_d, o_q);

   // parameters - none

   // module inputs
   input             clk;
   input             i_ld;
   input             i_clr;
   input [2:0]       i_d;

   // module outputs
   output [2:0]      o_q;

   // internal registers - none

   // output registers
   reg [2:0]         o_q;

   always @ (posedge clk)
   begin
      if(i_clr)
         o_q <= 0;
      else if(i_ld)
         o_q <= i_d;
   end
endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_ROMController.v                                                                            *
*   Description:    Organises access to all ROM                                                                  *
*   Date:           19/11/2004 20:57                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          27/11/2004 14:36 - First successful compile                                                  *
*                                                                                                                *
*****************************************************************************************************************/

/*****************************************************************************************************************
*                                                                                                                *
*   ROM Memory Map:                                                                                              *
*                                                                                                                *
*       (512KB = 2^19 bytes => 19 address bits => 5 hex digits)                                                  *
*                                                                                                                *
*       Address     Size    Description                                                                          *
*                                                                          00000      1                          *
*       00000       00800   Theta -> X,Y inc (byte 1)                                                            *
*       00800       00800   Theta -> X,Y inc (byte 2)                                                            *
*       01000       00800   Theta -> X,Y inc (byte 3)                                                            *
*                                                                                                                *
*       02000       01000   Square -> horizontal width (delta theta)                                             *
*       03000       01000   Square -> root (bits 12-4)                                                           *
*  ------------------------------------------------                        00001      2                          *
*       04000       01000   Square -> height on screen                                                           *
*       05000       01000   Square -> height scale                                                               *
*       06000       01000   Square -> intensity                                                                  *
*                                                                                                           *
*       07000       01000   Denom -> Div (bits 12-4)                                                        *
*  -------------------------------------------------                  00010       3                         *
*       08000       01000   Denom -> Div (bits 3-0)                                                         *
*                                                                                                           *
*       09000       01000   OppAdjRatio -> Theta (bits 11-3)                                                *
*       0A000       01000   OppAdjRatio -> Theta (bits 2-0)                                                 *
*                                                                                                           *
*       0B000       01000   Square -> root (bits 3-0)                                                       *
*  -------------------------------------------------                  00011       4                         *
*                                                                                                           *
*  -------------------------------------------------                  00100       5                         *
*       10000       01000   Texture 00                                                                      *
*       ...         ...     ...                                                                             *
*  -------------------------------------------------                  00101       6                         *
*       ...         ...     ...                                                                             *
*       17000       01000   Texture 07                                                                      *
*  -------------------------------------------------                  00110       7                         *
*       18000       01000   Texture 08                                                                      *
*       ...         ...     ...                                                                             *
*  -------------------------------------------------                  00111       8                         *
*       ...         ...     ...                                                                             *
*       1F000       01000   Texture 15                                                                      *
*  -------------------------------------------------                  01000       9                         *
*       20000       01000   Map 00                                                                          *
*       ...         ...     ...                                                                             *
*  -------------------------------------------------                  01001       10                        *
*       ...         ...     ...                                                                             *
*       27000       01000   Map 07                                                                          *
*  -------------------------------------------------                  01010       11                        *
*       28000       01000   Map 08                                                                          *
*       ...         ...     ...                                                                             *
*  -------------------------------------------------                  01011       12                        *
*       ...         ...     ...                                                                             *
*       2F000       01000   Map 15                                                                          *
*                                                                                                           *
*                                                                                                           *
************************************************************************************************************/

module m_ROMController(clk, reset, i_Angle, i_SquareValue, i_MapID, i_MapX, i_MapY, i_MapBlockSideSelect,
                       i_TextureID, i_TextureU, i_TextureV, i_Denominator, i_OppAdjRatio, i_ROMData,
                       i_ItemSelect, i_Load, o_AngleXChange, o_AngleYChange, o_SquareRootValue,
                       o_IntensityValue, o_ScreenHeight, o_HeightScale, o_MapTextureID,
                       o_ColourID, o_DivisionConstant, o_Arctangent, o_ROMAddress);

    // Parameters

    parameter p_WIDTH_ANGLE             = (11 - 1);
    parameter p_WIDTH_INCREMENT         = (12 - 1);

    parameter p_WIDTH_TEXTUREID         = (4  - 1);
    parameter p_WIDTH_TEXTURECOORDINATE = (6  - 1);
    parameter p_WIDTH_COLOURID          = (8  - 1);

    parameter p_WIDTH_MAPID             = (4  - 1);
    parameter p_WIDTH_MAPCOORDINATE     = (6  - 1);

    parameter p_WIDTH_SQUARE            = (12 - 1);
    parameter p_WIDTH_DEPTH             = (12 - 1);
    parameter p_WIDTH_INTENSITY         = (8  - 1);
//  parameter p_WIDTH_DTHETA            = (8  - 1);
    parameter p_WIDTH_HEIGHT            = (8  - 1);
    parameter p_WIDTH_HSCALE            = (8  - 1);

    parameter p_WIDTH_DENOMINATOR       = (12 - 1);
    parameter p_WIDTH_DIVISOR           = (12 - 1);

    parameter p_WIDTH_OARATIO           = (12 - 1);
    parameter p_WIDTH_ARCTAN            = (11 - 1);

    parameter p_WIDTH_ITEMSELECT        = (3  - 1);
    parameter p_WIDTH_BUSDATA           = (8  - 1);
    parameter p_WIDTH_ADDRESS           = (19 - 1);

    parameter p_WIDTH_STATE             = (4  - 1);
    parameter p_WIDTH_TWOINC            = (24 - 1);

    parameter p_STATE_ARCTAN1           = 00;
    parameter p_STATE_THETA1            = 01;
    parameter p_STATE_MAP               = 02;
    parameter p_STATE_TEXTURE           = 03;
//  parameter p_STATE_DTHETA            = 04; - OBSOLETE
    parameter p_STATE_SQRT1             = 05;
    parameter p_STATE_DIVIDE1           = 06;

    parameter p_STATE_IDLE              = 07;
    parameter p_STATE_THETA2            = 08;
    parameter p_STATE_THETA3            = 09;
    parameter p_STATE_HEIGHT            = 10;
    parameter p_STATE_HSCALE            = 11;
    parameter p_STATE_INTENSITY         = 12;
    parameter p_STATE_ARCTAN2           = 13;
    parameter p_STATE_DIVIDE2           = 14;
    parameter p_STATE_SQRT2             = 15;

    // Inputs

    input [p_WIDTH_ANGLE:0]             i_Angle;

    input [p_WIDTH_SQUARE:0]            i_SquareValue;

    input [p_WIDTH_MAPID:0]             i_MapID;
    input [p_WIDTH_MAPCOORDINATE:0]     i_MapX;
    input [p_WIDTH_MAPCOORDINATE:0]     i_MapY;
    input                               i_MapBlockSideSelect;

    input [p_WIDTH_TEXTUREID:0]         i_TextureID;
    input [p_WIDTH_TEXTURECOORDINATE:0] i_TextureU;
    input [p_WIDTH_TEXTURECOORDINATE:0] i_TextureV;

    input [p_WIDTH_DENOMINATOR:0]       i_Denominator;
    input [p_WIDTH_OARATIO:0]           i_OppAdjRatio;

    input [p_WIDTH_BUSDATA:0]           i_ROMData;

    input [p_WIDTH_ITEMSELECT:0]        i_ItemSelect;
    input                               i_Load;

    input                               clk;
    input                               reset;

    // Outputs

    output [p_WIDTH_INCREMENT:0]        o_AngleXChange;
    output [p_WIDTH_INCREMENT:0]        o_AngleYChange;

    output [p_WIDTH_DEPTH:0]            o_SquareRootValue;
    output [p_WIDTH_INTENSITY:0]        o_IntensityValue;
//  output [p_WIDTH_DTHETA:0]           o_DeltaTheta;
    output [p_WIDTH_HEIGHT:0]           o_ScreenHeight;
    output [p_WIDTH_HSCALE:0]           o_HeightScale;

    output [p_WIDTH_TEXTUREID:0]        o_MapTextureID;

    output [p_WIDTH_COLOURID:0]         o_ColourID;

    output [p_WIDTH_DIVISOR:0]          o_DivisionConstant;
    output [p_WIDTH_ARCTAN:0]           o_Arctangent;

    output [p_WIDTH_ADDRESS:0]          o_ROMAddress;

    // Bidirectionals - none
    // Wires - none

    // Internal Registers

    reg [p_WIDTH_STATE:0]               r_State;
    reg [p_WIDTH_TWOINC:0]              r_Increments;

    // Output Registers

    reg [p_WIDTH_DEPTH:0]               o_SquareRootValue;
    reg [p_WIDTH_INTENSITY:0]           o_IntensityValue;
//  reg [p_WIDTH_DTHETA:0]              o_DeltaTheta;
    reg [p_WIDTH_HEIGHT:0]              o_ScreenHeight;
    reg [p_WIDTH_HSCALE:0]              o_HeightScale;

    reg [p_WIDTH_TEXTUREID:0]           o_MapTextureID;

    reg [p_WIDTH_COLOURID:0]            o_ColourID;

    reg [p_WIDTH_DIVISOR:0]             o_DivisionConstant;
    reg [p_WIDTH_ARCTAN:0]              o_Arctangent;

    reg [p_WIDTH_ADDRESS:0]             o_ROMAddress;


    // Synchronous Block

    always @ (posedge clk)
    begin

        // Read in the current value

        r_Increments        = { (r_State == p_STATE_THETA1)     ? i_ROMData         : r_Increments[23:16],
                                (r_State == p_STATE_THETA2)     ? i_ROMData         : r_Increments[15:8],
                                (r_State == p_STATE_THETA3)     ? i_ROMData         : r_Increments[7:0] };

//      o_DeltaTheta        =   (r_State == p_STATE_DTHETA)     ? i_ROMData         : o_DeltaTheta;
        o_SquareRootValue   = { (r_State == p_STATE_SQRT1)      ? i_ROMData         : o_SquareRootValue[11:4],
                                (r_State == p_STATE_SQRT2)      ? i_ROMData[3:0]    : o_SquareRootValue[3:0] };
        o_ScreenHeight      =   (r_State == p_STATE_HEIGHT)     ? i_ROMData         : o_ScreenHeight;
        o_HeightScale       =   (r_State == p_STATE_HSCALE)     ? i_ROMData         : o_HeightScale;
        o_IntensityValue    =   (r_State == p_STATE_INTENSITY)  ? i_ROMData         : o_IntensityValue;

        o_DivisionConstant  = { (r_State == p_STATE_DIVIDE1)    ? i_ROMData         : o_DivisionConstant[11:4],
                                (r_State == p_STATE_DIVIDE2)    ? i_ROMData[3:0]    : o_DivisionConstant[3:0] };

        o_Arctangent        = { (r_State == p_STATE_ARCTAN1)    ? i_ROMData         : o_Arctangent[10:3],
                                (r_State == p_STATE_ARCTAN2)    ? i_ROMData[2:0]    : o_Arctangent[2:0] };

        o_ColourID          =   (r_State == p_STATE_TEXTURE)    ? i_ROMData         : o_ColourID;

        o_MapTextureID = (r_State != p_STATE_MAP) ? o_MapTextureID : i_MapBlockSideSelect ? i_ROMData[3:0] : i_ROMData[7:4];

        // Set the new state

        case (r_State)
            p_STATE_IDLE:       r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            p_STATE_THETA1:     r_State = reset ? p_STATE_IDLE : p_STATE_THETA2;
            p_STATE_THETA2:     r_State = reset ? p_STATE_IDLE : p_STATE_THETA3;
            p_STATE_THETA3:     r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            p_STATE_MAP:        r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            p_STATE_TEXTURE:    r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

//          p_STATE_DTHETA:     r_State = reset ? p_STATE_IDLE : p_STATE_SQRT1;
            p_STATE_SQRT1:      r_State = reset ? p_STATE_IDLE : p_STATE_SQRT2;
            p_STATE_SQRT2:      r_State = reset ? p_STATE_IDLE : p_STATE_HEIGHT;
            p_STATE_HEIGHT:     r_State = reset ? p_STATE_IDLE : p_STATE_HSCALE;
            p_STATE_HSCALE:     r_State = reset ? p_STATE_IDLE : p_STATE_INTENSITY;
            p_STATE_INTENSITY:  r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            p_STATE_DIVIDE1:    r_State = reset ? p_STATE_IDLE : p_STATE_DIVIDE2;
            p_STATE_DIVIDE2:    r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            p_STATE_ARCTAN1:    r_State = reset ? p_STATE_IDLE : p_STATE_ARCTAN2;
            p_STATE_ARCTAN2:    r_State = reset ? p_STATE_IDLE : i_Load ? {1'b0, i_ItemSelect} : p_STATE_IDLE;

            default:            r_State = p_STATE_IDLE;
        endcase

        // Set the address

        case (r_State)
            p_STATE_IDLE:       o_ROMAddress = 0;

            p_STATE_THETA1:     o_ROMAddress = {8'b00000000, i_Angle};
            p_STATE_THETA2:     o_ROMAddress = {8'b00000001, o_ROMAddress[10:0]};
            p_STATE_THETA3:     o_ROMAddress = {8'b00000010, o_ROMAddress[10:0]};

//          p_STATE_DTHETA:     o_ROMAddress = {7'b0000010, i_SquareValue};
            p_STATE_SQRT1:      o_ROMAddress = {7'b0000011, i_SquareValue};
            p_STATE_SQRT2:      o_ROMAddress = {7'b0001011, o_ROMAddress[11:0]};
            p_STATE_HEIGHT:     o_ROMAddress = {7'b0000100, o_ROMAddress[11:0]};
            p_STATE_HSCALE:     o_ROMAddress = {7'b0000101, o_ROMAddress[11:0]};
            p_STATE_INTENSITY:  o_ROMAddress = {7'b0000110, o_ROMAddress[11:0]};

            p_STATE_DIVIDE1:    o_ROMAddress = {7'b0000111, i_Denominator};
            p_STATE_DIVIDE2:    o_ROMAddress = {7'b0001000, o_ROMAddress[11:0]};

            p_STATE_ARCTAN1:    o_ROMAddress = {7'b0001001, i_OppAdjRatio};
            p_STATE_ARCTAN2:    o_ROMAddress = {7'b0001010, o_ROMAddress[11:0]};

            p_STATE_TEXTURE:    o_ROMAddress = {3'b001, i_TextureID, i_TextureU, i_TextureV};

            p_STATE_MAP:        o_ROMAddress = {3'b010, i_MapID, i_MapX, i_MapY};

            default:            o_ROMAddress = 0;
        endcase
    end

    // Assign increment outputs

    assign o_AngleXChange = r_Increments[23:12];
    assign o_AngleYChange = r_Increments[11:0];
endmodule

/*******************************************************************************************************************
*                                                                                                                  *
*   Filename:       m_ScreenXToTextrueU.v                                                                          *
*   Description:    Converts screen y coordinates into vertical texture coordinates                                *
*   Date:           27/11/2004 20:22                                                                               *
*   Author:         Robert Cassels                                                                                 *
*   Notes:          27/11/2004 22:07 - Frist seccessful compile                                                    *
*                                                                                                                  *
*******************************************************************************************************************/

module m_ScreenXToTextrueU(i_Depth, i_ScreenTheta, i_DirectionTheta, o_TextureU, o_CoordinateValid);

    // Parameters

    parameter p_WIDTH_DEPTH             = (12 - 1);
    parameter p_WIDTH_THETA             = (11 - 1);
    parameter p_WIDTH_VCOORDINATE       = (6  - 1);

    // Inputs

    input [p_WIDTH_DEPTH:0]             i_Depth;
    input [p_WIDTH_THETA:0]             i_ScreenTheta;
    input [p_WIDTH_THETA:0]             i_DirectionTheta;

    // Outputs
    output [p_WIDTH_VCOORDINATE:0]      o_TextureU;
    output                              o_CoordinateValid;

    // Bidirectionals- None

    // Wires

    wire [7:0]                          w_Useless1;
    wire [9:0]                          w_Overflow;
    wire [4:0]                          w_TexIntermed;
    wire [9:0]                          w_OFPlusOne;
    wire [p_WIDTH_THETA:0]              w_DeltaTheta;

    // Assign the angular difference

    assign w_DeltaTheta                 = i_ScreenTheta - i_DirectionTheta;

    // Instantiation

    m_multiplier12x11s mult(.a(i_Depth), .b(w_DeltaTheta), .o({w_Overflow, w_TexIntermed, w_Useless1}));

    // Internal Registers - None

    // Output Registers - None

    // Synchronous Block - None

    // Assignments

    assign w_OFPlusOne          = w_Overflow + 1;
    assign o_CoordinateValid    = (w_OFPlusOne[9:1] == 0) ? 1 : 0;
    assign o_TextureU           = {w_OFPlusOne[0], w_TexIntermed};

endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_ScreenYToTextrueV.v                                                                        *
*   Description:    Converts screen y coordinates into vertical texture coordinates                              *
*   Date:           27/11/2004 14:59                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          27/11/2004 15:34 - Frist successful compile                                                  *
*                                                                                                                *
*****************************************************************************************************************/

module m_ScreenYToTextrueV(i_HeightScale, i_Height, i_ScreenY, o_TextureV, o_CoordinateValid);

    // Parameters

    parameter p_WIDTH_HSCALE            = (8  - 1);
    parameter p_WIDTH_HEIGHT            = (8  - 1);
    parameter p_WIDTH_YCOORDINATE       = (8  - 1);
    parameter p_WIDTH_VCOORDINATE       = (6  - 1);

    // Inputs

    input [p_WIDTH_HSCALE:0]            i_HeightScale;
    input [p_WIDTH_HEIGHT:0]            i_Height;
    input [p_WIDTH_YCOORDINATE:0]       i_ScreenY;

    // Outputs

    output [p_WIDTH_VCOORDINATE:0]      o_TextureV;
    output                              o_CoordinateValid;

    // Bidirectionals- None

    // Wires

    wire [1:0]                          w_Useless1;
    wire [7:0]                          w_Useless2;
    wire [p_WIDTH_YCOORDINATE:0]        w_DistanceFromCentre;
    wire [p_WIDTH_YCOORDINATE:0]        w_DistanceFromCentreAbs;
    wire [p_WIDTH_YCOORDINATE:0]        w_DistanceFromBottom;

    // Instantiation

    m_cmple8 compare_less_equal(.A(w_DistanceFromCentreAbs), .B(i_Height), .A_LE_B(o_CoordinateValid));

    m_multiplier8x8 mult(.a(w_DistanceFromBottom), .b(i_HeightScale), .o({w_Useless2, o_TextureV, w_Useless1}));

    // Internal Registers - None

    // Output Registers - None

    // Synchronous Block - None

    // Assignments

    assign w_DistanceFromCentre = (i_ScreenY - 120);
    assign w_DistanceFromCentreAbs = w_DistanceFromCentre[7] ? -w_DistanceFromCentre : w_DistanceFromCentre;
    assign w_DistanceFromBottom = i_Height + w_DistanceFromCentre;
endmodule

/*****************************************************************************************************************
*                                                                                                                *
*   Filename:       m_SingleIncrementRegister.v                                                                  *
*   Description:    Holds and increments a value aligned to a wall boundary by adding 64 to the aligned value   *
*                   every time the increment signal is asserted.                                                 *
*   Date:           11/11/2004 10:51                                                                             *
*   Author:         Robert Cassels                                                                               *
*   Notes:          11/11/2004 11:18 - First successful compile                                                 *
*                                                                                                                *
*****************************************************************************************************************/

module m_SingleIncrementRegister(clk, i_InitialCoordinate, i_DirectionalIncrement,
                            i_LoadInitialValue, i_Increment, o_AlignedCoordinate);

    // Parameters

    parameter p_WIDTH_MAPCOORDINATE     = (12 - 1);
    parameter p_WIDTH_WALLCOORDINATE    = (6  - 1);

    // Inputs

    input [p_WIDTH_WALLCOORDINATE:0]    i_InitialCoordinate;
    input                               i_DirectionalIncrement;
    input                               i_LoadInitialValue;
    input                               i_Increment;

    input                               clk;

    // Outputs

    output [p_WIDTH_MAPCOORDINATE:0]    o_AlignedCoordinate;

    // Bidirectionals - none

    // Wires - none

    // Internal Registers

    reg [p_WIDTH_WALLCOORDINATE:0]      r_GranularCoordinate;

    reg [p_WIDTH_WALLCOORDINATE:0]      r_InitialValue;
    reg [p_WIDTH_WALLCOORDINATE:0]      r_RunningValue;

    // Output Registers - none

    // Synchronous Block

    always @ (posedge clk)
    begin
```

```
        // Calculate intermediate values

        r_InitialValue = i_DirectionalIncrement ? i_InitialCoordinate : (i_InitialCoordinate + 1);
        r_RunningValue = i_Increment ?  (i_DirectionalIncrement ? (r_GranularCoordinate - 1) :
                                        (r_GranularCoordinate + 1)) : r_GranularCoordinate;

        // Set the new granular coordinate

        r_GranularCoordinate = i_LoadInitialValue ? r_InitialValue : r_RunningValue;

    end

    assign o_AlignedCoordinate = {r_GranularCoordinate, 6'b0};
endmodule

/*****************************************************************************************************************
*                                                                                                               *
*   Filename:       m_VideoMemoryAccessController.v                                                             *
*   Description:    Contols all Video Memory accesses.                                                          *
*   Date:           10/11/2004 00:27                                                                            *
*   Author:         Robert Cassels                                                                              *
*   Notes:          11/11/2004 03:09 - First successful compile                                                 *
*                   23/11/2004 16:29 - Changed to registered read output                                        *
*                   07/12/2004 02:03 - Read was one cycle too early and caused incorrect values to be read      *
*                                                                                                               *
*****************************************************************************************************************/

module m_VideoMemoryAccessController(clk, reset, i_WriteColourID, i_WriteIntensity, i_WriteX,
                                    i_WriteY, i_WriteStart, i_ReadX, i_ReadY, i_ReadStart,
                                    i_HighLowUseSwap, o_WriteBusy, o_ReadColourID,
                                    o_ReadIntensity, o_ReadBusy, o_MemoryAddress, o_OutputEnable,
                                    o_WriteEnable, b_DataBus);

    // Parameters

    parameter p_WIDTH_COLOURID          = (8  - 1);
    parameter p_WIDTH_INTENSITY         = (8  - 1);
    parameter p_WIDTH_XCOORDINATE       = (9  - 1);
    parameter p_WIDTH_YCOORDINATE       = (8  - 1);
    parameter p_WIDTH_STATE             = (2  - 1);
    parameter p_WIDTH_DATA              = (16 - 1);
    parameter p_WIDTH_ADDRESS           = (18 - 1);

    parameter p_STATE_IDLE              = 0;
    parameter p_STATE_READ              = 1;
    parameter p_STATE_WRITEA            = 2;
    parameter p_STATE_WRITEB            = 3;

    // Inputs

    input [p_WIDTH_COLOURID:0]          i_WriteColourID;
    input [p_WIDTH_INTENSITY:0]         i_WriteIntensity;
    input [p_WIDTH_XCOORDINATE:0]       i_WriteX;
    input [p_WIDTH_YCOORDINATE:0]       i_WriteY;
    input                               i_WriteStart;

    input [p_WIDTH_XCOORDINATE:0]       i_ReadX;
    input [p_WIDTH_YCOORDINATE:0]       i_ReadY;
    input                               i_ReadStart;

    input                               i_HighLowUseSwap;

    input                               clk;
    input                               reset;

    // Outputs

    output                              o_WriteBusy;

    output [p_WIDTH_COLOURID:0]         o_ReadColourID;
    output [p_WIDTH_INTENSITY:0]        o_ReadIntensity;
    output                              o_ReadBusy;

    output [p_WIDTH_ADDRESS:0]          o_MemoryAddress;
    output                              o_OutputEnable;
    output                              o_WriteEnable;

    // Bidirectionals

    inout [p_WIDTH_DATA:0]              b_DataBus;

    // Wires - none

    // Internal Registers

    reg                                 r_ImpendingRead;
    reg                                 r_ImpendingWrite;
    reg [p_WIDTH_STATE:0]               r_State;
    reg                                 r_ReadUsingHigh;

    reg                                 r_Reading;
    reg                                 r_Writing;

    // Output Registers

    reg                                 o_WriteBusy;

    reg [p_WIDTH_COLOURID:0]            o_ReadColourID;
    reg [p_WIDTH_INTENSITY:0]           o_ReadIntensity;
    reg                                 o_ReadBusy;

    reg [p_WIDTH_ADDRESS:0]             o_MemoryAddress;
    reg                                 o_OutputEnable;
    reg                                 o_WriteEnable;

    // Synchronous Block

    always @ (posedge clk)
    begin

        // Assign the data outputs to the Video Interface

        {o_ReadColourID, o_ReadIntensity} = r_Reading ? b_DataBus : {o_ReadColourID, o_ReadIntensity};

        // Update chip selector register

        r_ReadUsingHigh = reset ? 0 : i_HighLowUseSwap ? (~r_ReadUsingHigh) : r_ReadUsingHigh;

        // Check Start signals & update Impending registers as needed

        r_ImpendingWrite = reset ? 0 : i_WriteStart ? 1 : r_ImpendingWrite;
        r_ImpendingRead  = reset ? 0 : i_ReadStart  ? 1 : r_ImpendingRead;

        // Set the new state

        case (r_State)
            p_STATE_IDLE:   r_State = reset ? p_STATE_IDLE : r_ImpendingRead  ? p_STATE_READ   :
                            r_ImpendingWrite ? p_STATE_WRITEA : p_STATE_IDLE;

            p_STATE_READ:   r_State = reset ? p_STATE_IDLE : r_ImpendingWrite ? p_STATE_WRITEA :
                            p_STATE_IDLE;

            p_STATE_WRITEA: r_State = reset ? p_STATE_IDLE : p_STATE_WRITEB;

            p_STATE_WRITEB: r_State = reset ? p_STATE_IDLE : r_ImpendingRead ? p_STATE_READ    :
                            p_STATE_IDLE;

            default:        r_State = p_STATE_IDLE;
        endcase

        // Set reading/writing registers

        r_Reading =  (r_State == p_STATE_READ)                                   ? 1 : 0;
```

```
        r_Writing = ((r_State == p_STATE_WRITEA) || (r_State == p_STATE_WRITEB)) ? 1 : 0;

        // Update impending signals based on state changes

        r_ImpendingWrite = r_Writing ? 0 : r_ImpendingWrite;
        r_ImpendingRead  = r_Reading ? 0 : r_ImpendingRead;

        // Set the busy signals

        o_WriteBusy = reset ? 0 : r_ImpendingWrite ? 1 : r_Writing ? 1 : 0;
        o_ReadBusy  = reset ? 0 : r_ImpendingRead  ? 1 : r_Reading ? 1 : 0;

        // Set memory write/output enables

        o_OutputEnable = ~(reset ? 0 : r_Reading);
        o_WriteEnable  = ~(reset ? 0 : (r_State == p_STATE_WRITEA) ? 1 : 0);

        // Calculate the memory address using the correct X and Y

        o_MemoryAddress = r_Writing ? {~r_ReadUsingHigh, i_WriteX, i_WriteY} : {r_ReadUsingHigh, i_ReadX, i_ReadY};

    end

    // Make the tri-state buffer

    assign b_DataBus = r_Writing ? {i_WriteColourID, i_WriteIntensity} : 16'bZ;

endmodule

/*******************************************************************************************************************
*                                                                                                                  *
*   Filename:       m_WallPositionDelay.v                                                                          *
*   Description:    Delays the wall coordinates by 1 clock cycle                                                   *
*   Date:           27/11/2004 19:20                                                                               *
*   Author:         Robert Cassels                                                                                 *
*   Notes:          None                                                                                           *
*                                                                                                                  *
*******************************************************************************************************************/

module m_WallPositionDelay(clk, i_Load, i_IsSide, i_WallPositionX, i_WallPositionY, o_WallPositionX,
                           o_WallPositionY, o_IsSide);

    // Parameters

    parameter p_WIDTH_COORDINATE        = (12 - 1);

    // Inputs

    input                               clk;

    input                               i_Load;

    input                               i_IsSide;
    input [p_WIDTH_COORDINATE:0]        i_WallPositionX;
    input [p_WIDTH_COORDINATE:0]        i_WallPositionY;

    // Outputs

    output                              o_IsSide;
    output [p_WIDTH_COORDINATE:0]       o_WallPositionX;
    output [p_WIDTH_COORDINATE:0]       o_WallPositionY;

    // Internal Registers

    reg                                 r_IsSide;
    reg [p_WIDTH_COORDINATE:0]          r_WallPositionX;
    reg [p_WIDTH_COORDINATE:0]          r_WallPositionY;

    // Output Registers

    reg                                 o_IsSide;
    reg [p_WIDTH_COORDINATE:0]          o_WallPositionX;
    reg [p_WIDTH_COORDINATE:0]          o_WallPositionY;

    // Synchronous Block

    always @ (posedge clk)
    begin

        // Make a registered delay

        r_WallPositionX <= i_Load ? i_WallPositionX : o_WallPositionX;
        r_WallPositionY <= i_Load ? i_WallPositionY : o_WallPositionY;
        r_IsSide        <= i_Load ? i_IsSide        : o_IsSide;

        o_WallPositionX <= i_Load ? r_WallPositionX : o_WallPositionX;
        o_WallPositionY <= i_Load ? r_WallPositionY : o_WallPositionY;
        o_IsSide        <= i_Load ? r_IsSide        : o_IsSide;
    end
endmodule
```