Project Report: CUDoom

CSEE4840 Embedded System Design
Columbia University, Spring 2013

Alden Goldstein	**(ag3287)**
Edward Garcia	**(ewg2115)**
Minyun Gu	**(mg3295)**
Wei Hao Yuan	**(wy2211)**
Yiming Xu	**(yx2213)**

Contents

Introduction

CUDoom is a project inspired by the Doom, one of the last video games to use ray casting techniques to create a pseudo 3D environment. CUDoom creates a similar 3D world and allows a player to freely move around it. Among the key features in CUDoom is the fact that the entire world is fully texture mapped to the player's perspective. Walls can be set to two different heights, the floor consists of tiles of different textures and the sky rotates to match the player's frame of view.

The project is divided into software and hardware components. The software keeps track of the player position and frame of view, accepts keyboard inputs and generates music for the world. Hardware consists of a ray casting accelerator and the logic needed to texture map the environment. Individual screen pixel calculations are generated on the fly and the project runs smoothly at 60 frames per second for a 640x480 screen.

System Overview

A high level overview of the system is provided in figure 1. The system is designed to update a 640x480 display at 60 frames per second using fixed point arithmetic. A phase locked loop (PLL) generates the following timings needed by the rest of the system: 50 MHz, 50MHz phase delayed, and 25 MHz.

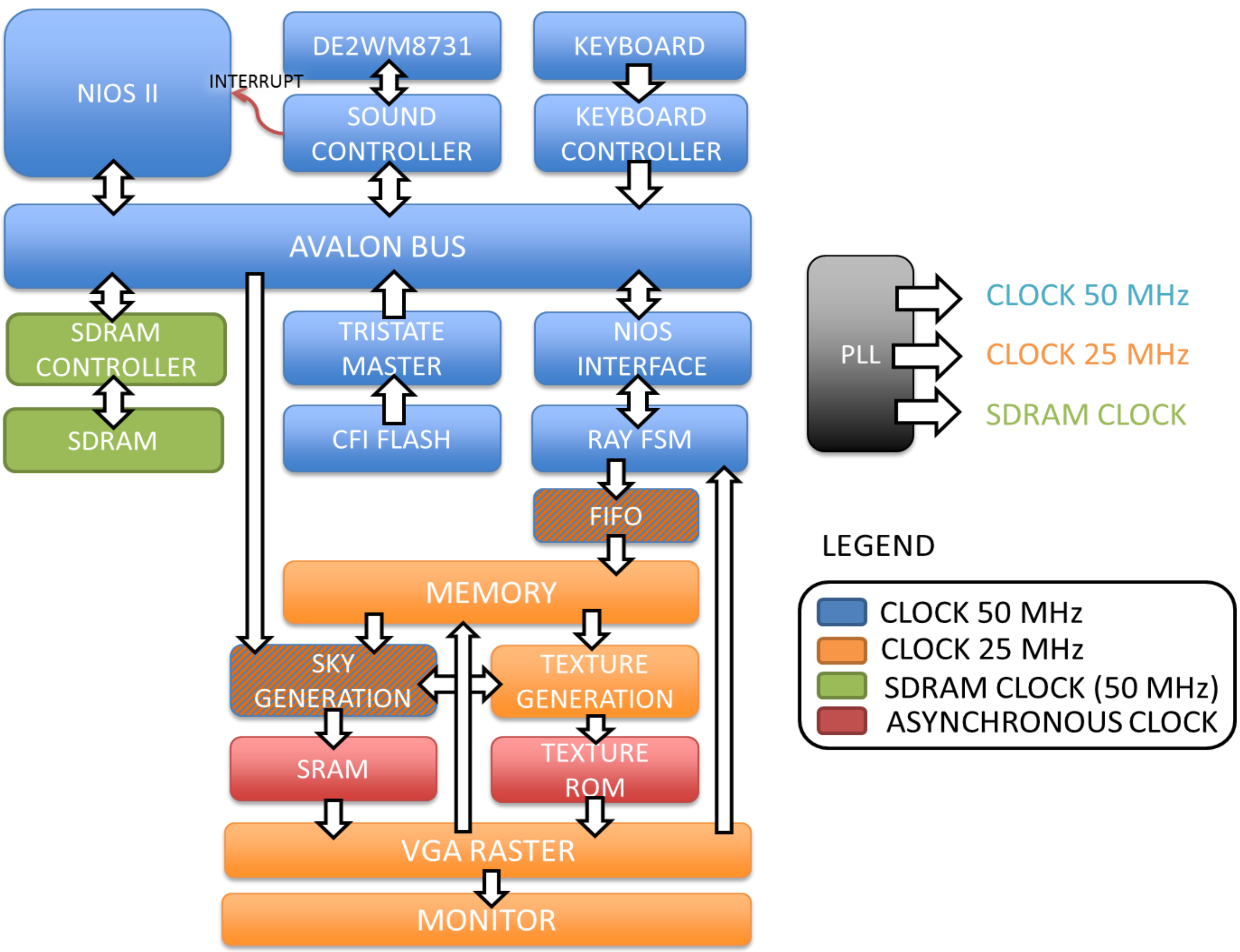


Figure 1. Overall System Architecture

Software Logic Overview

The NIOS II processor and the peripherals that it interacts with run at 50 MHz. The only exception is the SDRAM which stores the program memory and runs on the 50 MHz phase delayed clock. These components work together and are controlled by the software portion of our project.

At a high level, the software performs the following tasks:

- The NIOS initializes the system by calculating and storing the sine and cosine tables necessary for future calculations.
- Also part of the initialization, the NIOS downloads the sky texture from the SDRAM into the SRAM on the board
- The NIOS polls the keyboard looking for an input from the user. If a key is detected, it will update the data concerning the player's position and frame of view.
- The NIOS keeps track of calculating a frame. For each frame, it will cast 640 rays for each column within the frame. The goal is to calculate the wall heights for each of the columns on the screen based on the respective distance of the wall to the player.
    - Each individual column calculation is hardware accelerated. The software passes the angle information of the respective ray to the Ray FSM hardware module through a simple handshake protocol.
- Throughout the software portion of the program, the sound controller sends interrupts to the processor whenever it needs the next note to play. During an interrupt, the next sample is fetched from flash memory and written to the sound controller.

Hardware Logic Overview

The VGA Raster module drives the rest of the hardware components that do not interact with the NIOS. Most of these components run at 25 MHz. At a high level the hardware performs the following tasks:

- The VGA Raster module signals the start of a new frame. This causes the VGA Raster and Ray FSM modules to swap the memory buffer locations that they are respectively reading and writing to.
    - The Ray FSM module computes wall heights for individual columns based on the player distance to the wall along the path of the ray specified by software. This and other intermediate variables are safely stored through the use of a FIFO and memory buffer.

- In parallel with the Ray FSM module, the VGA module begins to read from its respective memory location. The memory outputs are intermediate variables that are fed into the Sky Generation and Texture Generation blocks.
    - The sky is calculated for the respective pixel. The Sky Generation module ensures the sky is mapped to match the x coordinate of the walls. An address is generated to pull the respective sky pixel from SRAM. The SRAM output is fed into the VGA raster component.
    - In concert with Sky Generation module the logic, the Texture Generation module maps the wall and floor textures of the pixel based on the vantage point of the player. An address is generated to pull the respective texture pixel from Texture ROM, a lookup table containing all the wall and floor textures. The Texture ROM output is fed into the VGA raster component.
- The VGA raster component selects the appropriate pixel stream and converts it to the representation needed by the monitor. Depending on the direction of the wall to the user, it will shade it appropriately.

Algorithm

World Map

The world map is represented by a 32x32 array where each value represents a cube in the world. Array entries that are 0 represent an empty space that the player can walk through. Array entries larger than 0 represent a cube with a specific texture. Table 1 gives the respective value of each texture.

```
char worldMap[mapWidth][mapHeight]=
{
  9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9 ,
  9,0,0,7,0,8,0,8,0,0,8,7,0,2,7,2,7,2,7,2,0,7,0,9 ,
  9,2,3,3,0,0,0,0,0,8,8,4,0,0,0,0,0,0,0,0,0,0,0,9 ,
  9,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,9 ,
  9,2,3,3,0,0,0,0,0,8,8,4,0,0,0,0,0,0,0,0,0,0,0,9 ,
  9,0,0,0,0,0,0,0,0,0,8,4,0,0,0,0,0,6,6,6,0,6,7,9 ,
  9,8,8,8,0,8,8,8,8,8,8,4,4,4,4,4,4,6,0,0,0,0,0,9 ,
  9,7,7,7,0,7,7,7,7,0,8,0,8,0,8,0,8,4,0,4,0,6,7,9 ,
  9,7,0,0,0,0,0,0,7,8,0,8,0,8,0,8,8,6,0,0,0,0,0,9 ,
  9,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,6,0,0,0,0,7,9 ,
  9,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,6,0,6,0,6,0,9 ,
  9,7,0,0,0,0,0,0,7,8,0,8,0,8,0,8,8,6,4,6,0,6,6,9 ,
  9,2,7,7,0,7,7,7,7,8,8,4,0,6,8,4,8,3,3,3,0,3,6,9 ,
  9,7,2,2,0,2,2,2,2,4,6,4,0,0,6,0,6,3,0,0,0,0,0,9 ,
  9,7,0,0,0,0,0,2,2,4,0,0,0,0,0,0,4,3,0,0,0,0,6,9 ,
  9,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,4,3,0,0,0,0,0,9 ,
```

```
  9,7,0,0,0,0,0,0,1,4,4,4,4,4,6,0,6,3,3,0,0,0,6,9 ,
  9,0,0,0,0,0,0,0,2,2,2,1,2,2,2,6,6,0,0,5,0,5,0,9 ,
  9,7,0,0,0,0,0,2,2,2,0,0,0,2,2,0,5,0,5,0,0,0,5,9 ,
  9,0,0,0,0,0,0,0,2,0,0,0,0,0,2,5,0,5,0,5,0,5,0,9 ,
  9,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9 ,
  9,0,0,0,0,0,0,0,2,0,0,0,0,0,2,5,0,5,0,5,0,5,0,9,
  9,2,7,2,7,2,7,2,2,2,0,7,0,8,8,0,5,0,5,0,7,0,5,9 ,
  9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
};
```

Value	Texture
0	Empty spaces
1 - 4	Textures 1 through 4 of normal height
5 - 8	textures 1 through 4 of taller height
9	fake wall, i.e. sky outlook.

Table 1. Texture Mappings

Ray Casting

The ray casting algorithm involves casting individual rays, only for each column. Due to the fixed perspective, there is very little calculation that needs to be done afterward once column parameters are calculated. Essentially, we determine the height of each wall by finding out how far we are from it. The main equation for ray casting is…

$$percieved\ column\ height = \frac{constant}{wall\ distance}$$

Other features, such as textures and floors, can be determined from a few additional calculations, but the fundamental algorithm remains as one of low complexity.

As a basis for our project, we drew majority of our resources from LodeV's Ray Casting tutorial [1]. We started with code from his website, which includes a C++ version with textures, floors, and ceilings. While his code was a great starting point and gave us an immediate working version to play around with, we had to make many modifications before we could port anything to hardware. To explain the changes we made, we have to explain the ray casting algorithm more in depth.

For ray casting, we need to increment a ray for each column to find a wall and determine the appropriate distance. The larger the ray increments are, the more inexact the measurements will be, and the more likely part of a wall will be missed. Reducing the increments will give finer edges, but at the expense of being significantly slower.

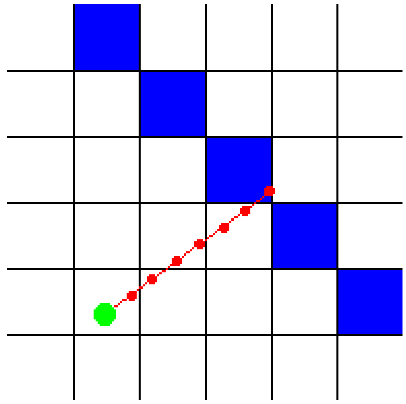


Figure 2. Fixed increment wall search

Image Credit: Lode's Computer Graphics Tutorial [1]

Pseudo-code for wall finding:

```
// initialize
ray position = current player position
distance = 0

while not hit wall
     increment ray position
     increment distance
```

This algorithm, while naïve, will place the "light" ray relatively close to the correct position.

LodeV uses a more sophisticated approach. Since the ray casting model we use only involves orthogonal walls on a map, finding walls can be done by hitting every edge. The algorithm used for this is called DDA, and is a modified version of Breshenham's line algorithm. Professor Edwards pretty quickly recognized that this approach could be used for Ray Casting. The C++ code from LodeV's website is given below.

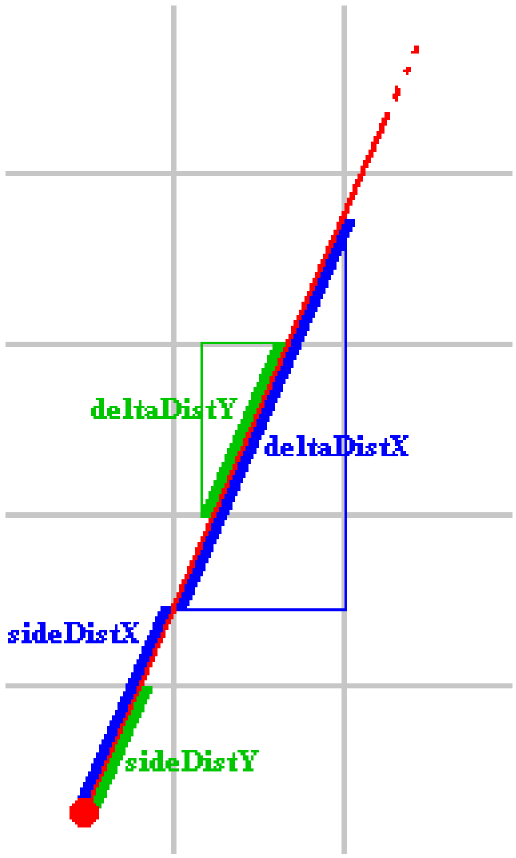


Figure 3. DDA

Image Credit: Lode's Computer Graphics Tutorial [1]

C++ code for DDA (given by LodeV)

```
//perform DDA
while (hit == 0)
{
       //jump to next map square, OR in x-direction, OR in
       y-direction
       if (sideDistX < sideDistY)
       {
              sideDistX += deltaDistX;
              mapX += stepX;
              side = 0;
```

```
        }
        else
        {
                sideDistY += deltaDistY;
                mapY += stepY;
                side = 1;
        }

        //Check if ray has hit a wall
        if (worldMap[mapX][mapY] > 0)
                hit = 1;
}
```

Drawbacks of DDA

In the normal iterative procedure, you are finding almost exactly where the ray hits the wall. In DDA you are finding which side has been hit. In a sense, the iterative procedure gives more information. If you reduce the bits for calculations on the iterative procedure, you have a soft failure, i.e. the errors get predictably worse with less precision. For DDA however, you are making a choice in each step of the loop. Essentially you are comparing whether you are closer to an x-wall or a y-wall. You can either be right or wrong, thus this carries a notion of hard failure, which can look erratic on the screen when you starting removing precision.

So in a sense, we upgraded by downgrading. We used fixed wall increments instead for more robustness and reliability. We wanted to make sure everything would work once we took the time make the actual system. On top of the fixed increment approach, we added another loop that back-traces in smaller increments. This helps makes the ray position more exact, with larger initial search increments (better speed). Of course, wall misses still occur at the same frequency.

Other Modifications from LodeV's algorithm

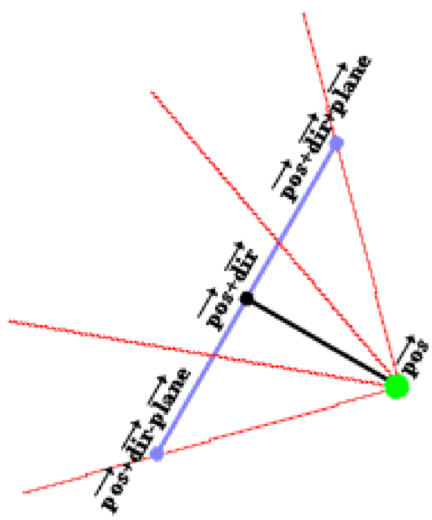


Figure 4. LodeV's camera plane method

Image Credit: Lode's Computer Graphics Tutorial [1]

LodeV employs a camera plane and a direction plane to find the ray vectors. We use fixed angles instead, which actually helps remove several multiplications, since we don't perform rotation matrix multiplication. Removing multiplications from the software step actually increases precision, since we have more room to avoid overflow (on 32 bit processor). We change direction by incrementing the index for lookup table. The cosine lookup table is the x-direction, the sinusoid lookup table is for the y-direction. Of course these could be combined into one lookup table, but there is no need to because we are using SDRAM.

In addition, this method, combined the iterative procedure, simplifies fish-eye effect correction calculations as well. By using fixed angles, we know our fish-eye angle (perpendicular distance from player to wall) will directly align with the lookup tables, and thus we can use $cosine[fish\_angle]/(increment\ factor)$ as our distance increment in the loop (since the increment factor is a multiple of 2, we actually use a bit shift).

Sky

We added a sky on top of the LodeV's Ray Casting version. The sky is more than just a fixed background picture. To give the illusion of movement, the sky's x-coordinates must change directly with those of the walls. In addition, the sky must appear circularly looped with itself. Not many pictures fit this requirement. Thus we had to resort to texture generation. Fortunately another portion (not ray casting) of LodeV's website had a random noise texture generation, that involves interpolation of finer and grainer random noise values. This gives the illusion of clouds. We had to modify the texture so the beginnings and ends were also interpolated with each other, to give the illusion of a circular buffer. This also means the sinusoid lookup tables have to have a size that is a multiple of the sky width (1024), so the beginning and end line up properly. All of this together gives the illusion of a huge, full sky that fits into the (relatively small) SRAM.

Multiple Heights

Another thing we added was multiple heights (2 different heights to be exact). We made this enhancement, since we knew it could be done easily in parallel with the original single height wall. Essentially, we cast two rays in parallel, one that stops at walls of all heights, and one that stops at walls of taller heights. Then we do very similar calculations on each wall. The drawing parameters are as follows:

drawStart = top of tall walls
drawMid = top of normal wall
draw End = bottom of either tall or normal wall

$$drawStart = mid\ screen - 5 \times column\ height/2$$
$$drawMid = mid\ screen - column\ height/2$$
$$drawEnd = mid\ screen + column\ height/2$$

We see that the height increase is a simply caused by multiplicative factor we add to the column height. The factor of 5 gives the illusion of being three times the height of a normal wall. At the texture generator, we use the values to multiplex whether to draw a floor, a normal wall texture, or a tall wall texture. At the VGA, we use "drawStart" and the fake walls to multiplex whether or not to draw a sky pixel or the texture pixel from the texture generator.

Textures

We generate textures almost identically to LodeV's code. LodeV actually uses a tangent angle to find textures (since he uses DDA), however because the increment method yields a nearly exact map position, we have more information than just a wall. We can easily find the x-coordinate of the texture pixel with a modulo-64 operation (64 is the width of the textures, as shown below). In additional little nuance to textures is that we must know whether we hit and x or a y-wall. In DDA this is given as the output. We however used a simple difference comparison. In other words, we did the following, shown in the pseudo code below.

```
If abs(current x position – closest x-wall) < abs(current y
position – closest y-wall)
     Choose x-wall
Else
     Choose y-wall
```

Knowing whether we hit an x or y-wall, also allowed us to choose whether or not to shade a wall. This gives a nice effect, when the x-walls are shaded and the y-walls are not (and vice versa). The world has a total of four textures, copied from the Wolfenstein 3D game. Each texture consists of 64x64 pixels with each pixel color represented by 24 bits. The textures are shown below

Figure 5. Textures used in project

We didn't really add or change much to the fundamentals of LodeV's texture and floor generation. We focused on small simplifications and how we would lay it out in hardware. We explained calculation of the x-component of the wall texture above. This can be done as a function of the ray position. The y-component is a little trickier. Essentially the values are calculated through interpolation between the drawStart and drawEnd. The floors use a slightly different algorithm, but nevertheless use interpolation from drawEnd to the bottom of the screen.

Hardware

Ray FSM

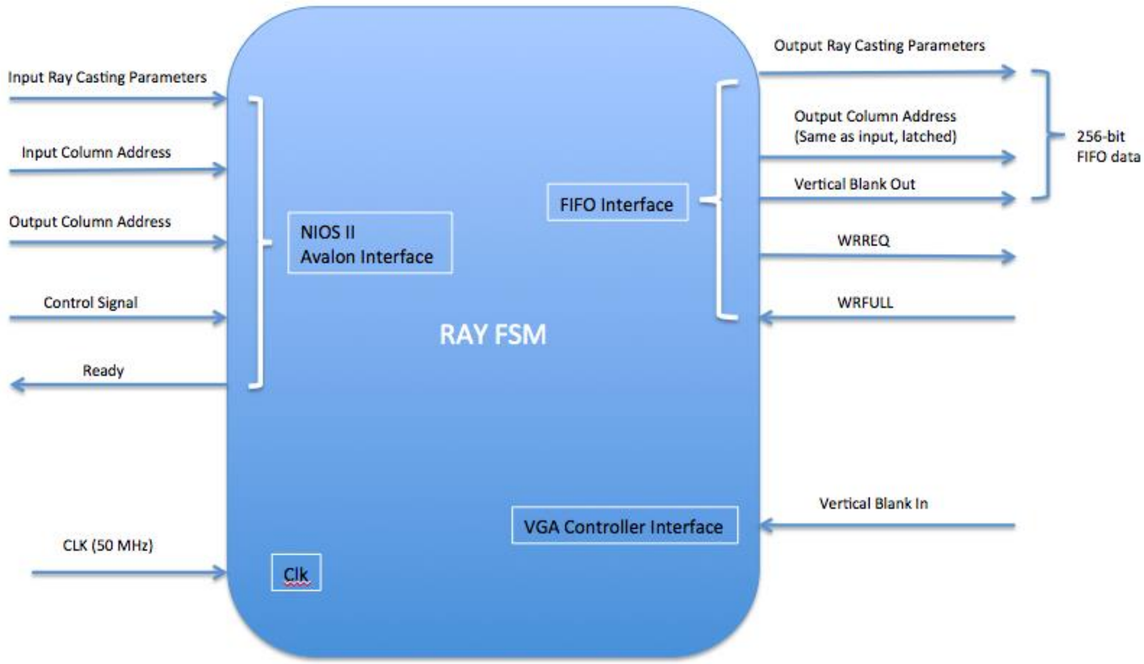


Figure 6. Ray FSM module

The motivation behind hardware acceleration for ray casting, mainly lies in the wall finding process. Using 1/32 of a square side increments over a 32 X 32 map, we have a worst case of:

$$32 \times 32 \times \sqrt{2} \approx 1500 \; iterations\ per\ column$$

$$1500 \; iterations\ per\ column \times 640 \; columns\ per\ screen \approx 1 \; million\ iterations\ per\ frame$$

If we use a software loop, this would have a wasteful, especially for calculations that are so simple. The inside of loop is essentially just increments, a memory fetch, and a comparison, all of which can be done easily within a 50MHz clock cycle. This is the reason for the Ray FSM module. The name says it all; it is a finite state machine for each column ray, i.e. it computes the common set of parameters for each column. Notice that after the increment stage, there is also a division stage. In addition, there is combinational logic that executes during the division stage, which yields the necessary parameters for floor and textures.

Besides the calculations, which are all part of the ray casting algorithm, the Ray FSM is important for its interaction with both the Nios II processor, and its interaction with the FIFO to feed the column memory. The diagram is shown in figure 6. When the Ray

FSM is in the ready state, it is basically telling software that it is ready for a new set of parameters to calculate. The software preloads the inputs through the Avalon Bus before the Ray FSM even asserts the 'ready' signal. However, these parameters don't get latched in until the software pulses a 'control' signal. This gives an efficient pipeline between software and hardware. Software can calculate and load new parameters while the Ray FSM is still working. Once the software sees the 'ready' signal, all it needs to do is pulse a single bit (in reality 32 bits, due to 32 bit Avalon Bus). Then the software can resume computing parameters for the next column.

Frame Synchronization

We want to explain frame synchronization briefly since the next two sections are related to this manner. Frame synchronization is achieved by using twice as much memory. This allows writes to one column memory, while the VGA computes from the other column memory. Only when a memory buffer is completely written to, the reads and writes are toggled, and the VGA will compute values new set of complete data. Thus, the VGA will always be reading complete data, and frame synchronization will be achieved.

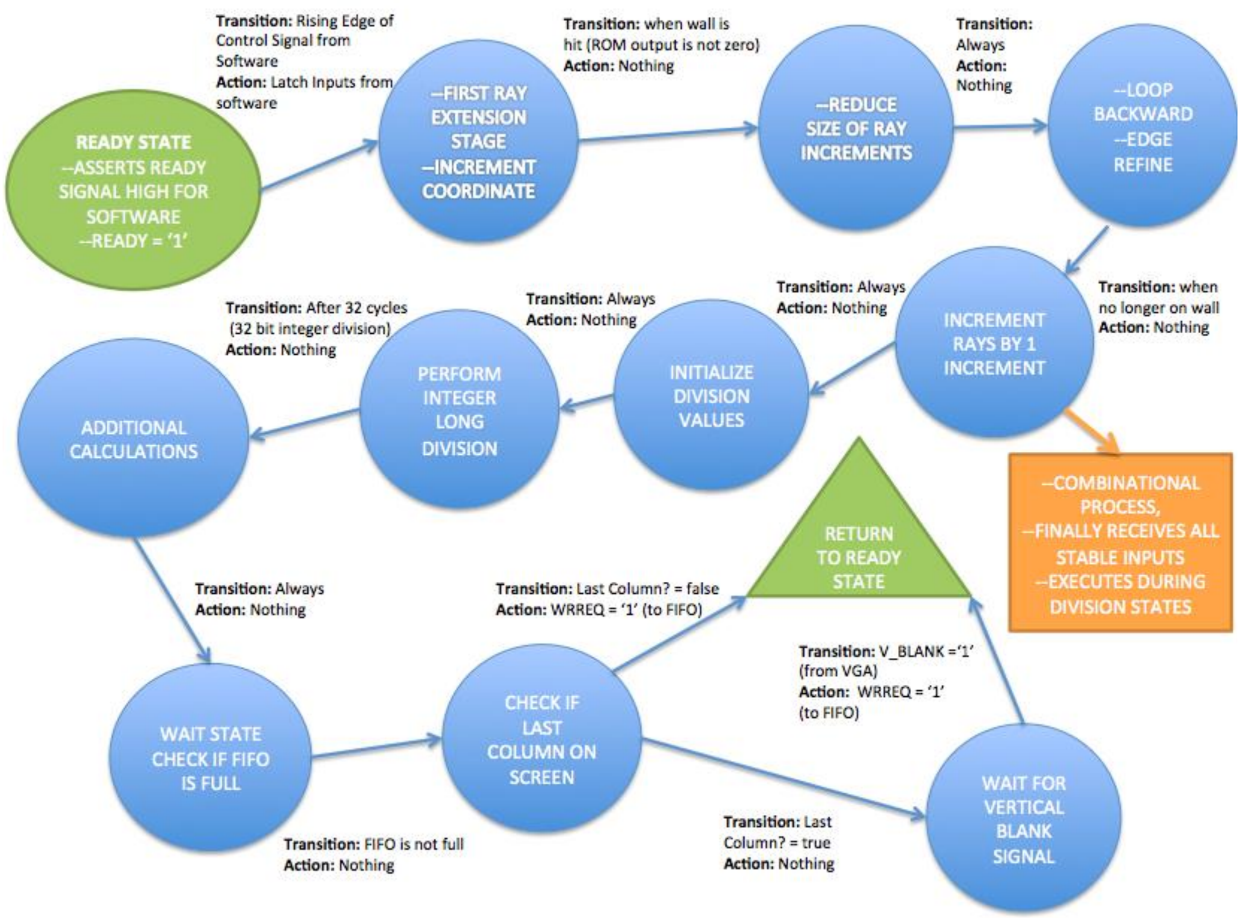


Figure 7. Ray FSM Overall State Machine

Foregoing a Frame Buffer

If it isn't clear by now, we reiterate the fact in ray casting; there are many parameters that are common to all pixels in a given column. This is in contrast to ray tracing, which traces a ray for every pixel on the screen (thus has unique parameters for every pixel). After a common set of column parameters are calculated, we can use only few pipelined calculations (RAM fetch + 2 cascaded multiplications + some additions + some MUXing + a ROM fetch) to give an appropriate texture pixel. Therefore a frame buffer is not necessary, and VGA pixel calculations can be done on the fly, once the column parameter set has been computed. Thus, for each column on the screen, a memory location was created to hold the set of necessary column parameters. The bit addresses for the column parameters are stored as follows

	# of bits	X down to	Y
2 bit gap	2	1	0
texX	6	7	2
drawEnd	9	16	8
drawStart	9	25	17
invline	18	43	26
line_minus_h	18	61	44
isSide2	1	62	62
isSide	1	63	63
floorX	18	81	64
floorY	18	99	82
tmpPosX	18	117	100
tmpPosY	18	135	118
invdist_out	12	147	136
12bit Gap	12	159	148
data_out	10	169	160
2 bit gap	2	171	170
texX2	6	177	172
drawMid	9	186	178
invLine2	18	204	187
line_minus_h2	18	222	205
bool	1	223	223
texNum	4	227	224
texNum2	4	231	228
colAddress	10	241	232
VGA_Blank	1	242	242

Table 2. Bit positions for column parameters inside memory and FIFO. Note that items in red appear in the FIFO and not the memory.

Memory

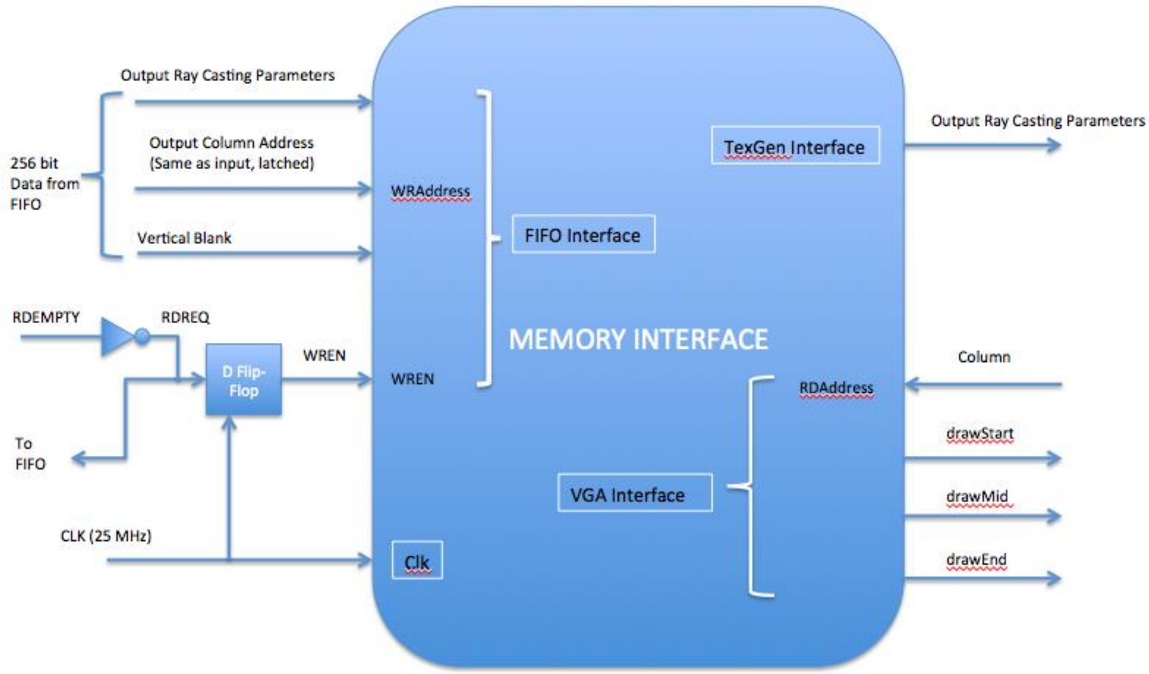


Figure 8. Memory module

As we explain above, the memory module will take in data from the FIFO that includes the column address. Thus the input port to the memory module we created does not even take in a column address. In addition, it also does not take in a write enable. Instead it uses the read request signal from the FIFO. As shown in the diagram above the RDREQ is simply the inverted RDEMPTY signal. Thus the memory module takes data from the FIFO whenever it is available. Notice that the vertical blank signal comes with the data as well.

The bulk of our memory is contained in an M4K Ram, which takes up almost the entire board. The pixel parameters require 212 bits. Since, we store memory in a column array, we need a 256-bit width for each column. There are 640 columns. If we use an M4K, we require block configurations in powers of two. Naively, we would have a 1024 by 256-bit column buffer. To add frame synchronization, we must have two memories that are toggled every time a VGA blank signal comes in.

$$1024 \times 256\ bits \times 2\ memories = 524{,}288\ bits$$

To make this fit on the board, we have two options:

A. Reduce bits. Cut down bits to 192, and splice a 128 bit by 1024 memory, with a 64 bit by 1024 memory. Requires 96 M4K blocks according to Megafunction Wizard.

B. Reduce number of addresses. Use a full 256 bits, but have a memory with 512 addresses and a memory with 128 addresses. Requires 74 M4K blocks according to Megafunction Wizard.

We opted for option B. This is because it offers a smaller amount of memory and requires no cutting of bits. Reducing the number bits to fit 192 bit memory size gives degradation of the image when tested in software. While 96 M4K's fit on the board along with the processor, we used the extra space on the board for the FIFO between clock domains.

Memory Issues

Option B, presented an interesting problem. We encountered strange error on the VGA glitch that seemed to occur right where we switch from the 512-address memory to the 128-address memory. While, the switching of writes and outputs were combinational, and timing requirements were well within acceptable range (approximately 5 ns setup slack). Even with this, there were still timing issues on the VGA, where it seems that the change between output buffers did not stabilize completely. To fix this, we used a "patch" memory block. This is illustrated in the diagram below. The original scheme switches reads and writes at the same location, i.e. the end of the 512-word memory, and the beginning of the 128-word memory. This caused a line on the screen directly at the memory switching point, no matter how much timing slack we had. Due to time constraints we had to come up with a rather crude fix, which was the patch memory block. As seen in Figure 9, writes for the patch block begin a few addresses from the edges of the memory (for extra safety) and bridge the connection between the two larger memory blocks.

The original scheme was to switch reads and writes at the same spot along the edges of the memory. In order to smooth transitions, we simultaneously write to a patch memory that overlaps with the other two memory blocks. We also see in the diagram, that we no longer need to write at the edges in the memory. The reason for this is because at the edge of the memory, the next address will give huge transition in bits (i.e. all 1's to all 0's for the address). In addition, we see that reads switch in different places than from the writes, thus when we switch read buffers, we are switching between two identical values. All of this combined, keeps the transitions of inputs and outputs minimal, and miraculously fixed our error!

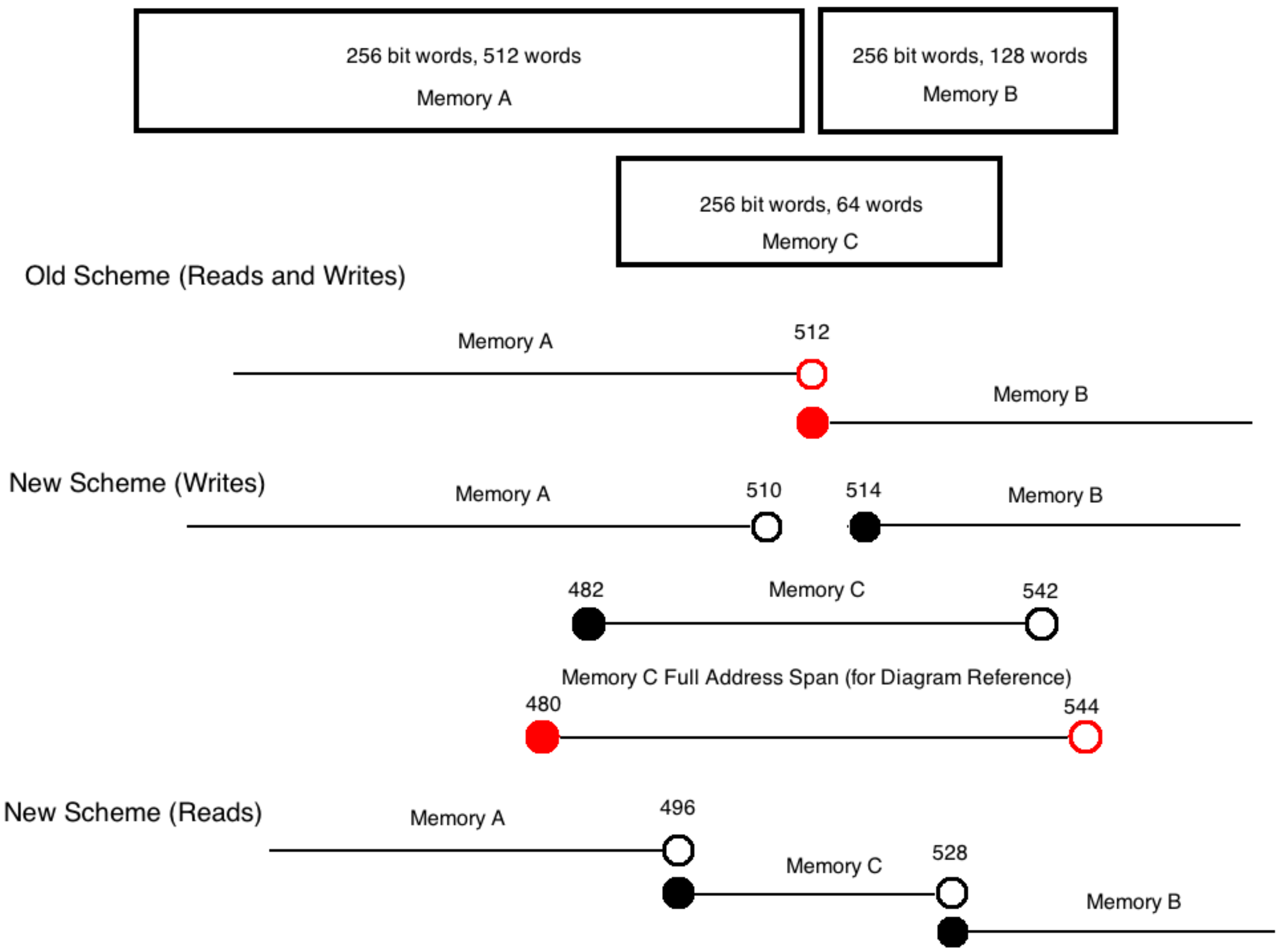


Figure 9. Overlapping memory modules

FIFO & PLL

The D-Flip-Flop generated 25 MHz clock seemed to give us some very strange glitches (we weren't sure what they were caused by at the time). In addition we also had system crashes that we believed were due to corruption of the M4K memory. First, with the help of our great TA (Luis), we were able to switch the generation of the 25 MHz clock to a PLL. This seemed to improve the screen glitches, but we still faced some occasional system crashes. Because we suspected M4K corruption, it was apparent that this was likely due to its interaction between two clocked domains (i.e. reads from 25 MHz domain, writes from 50 MHz domain). Before we added a FIFO, only the VGA was running on 25 MHz, and would feed read addresses directly to the M4K column memory module that we mentioned above. To solve the issues, we decided to organize clock domains and make everything from the memory module through to the VGA controller as part of a single clocked domain (25 MHz). This meant making texture generation as part of the 25 MHz domain as well (this actually works out nicely considering the critical path in this module does not fit a 50 MHz domain). However, the Ray FSM was still writing to memory. Since the Ray FSM runs at 50 MHz (this is necessary for speed), we placed a FIFO between the Ray FSM and the memory module, essentially creating a safe bridge between the 50 MHz domain and the 25 MHz domain.

VGA Raster

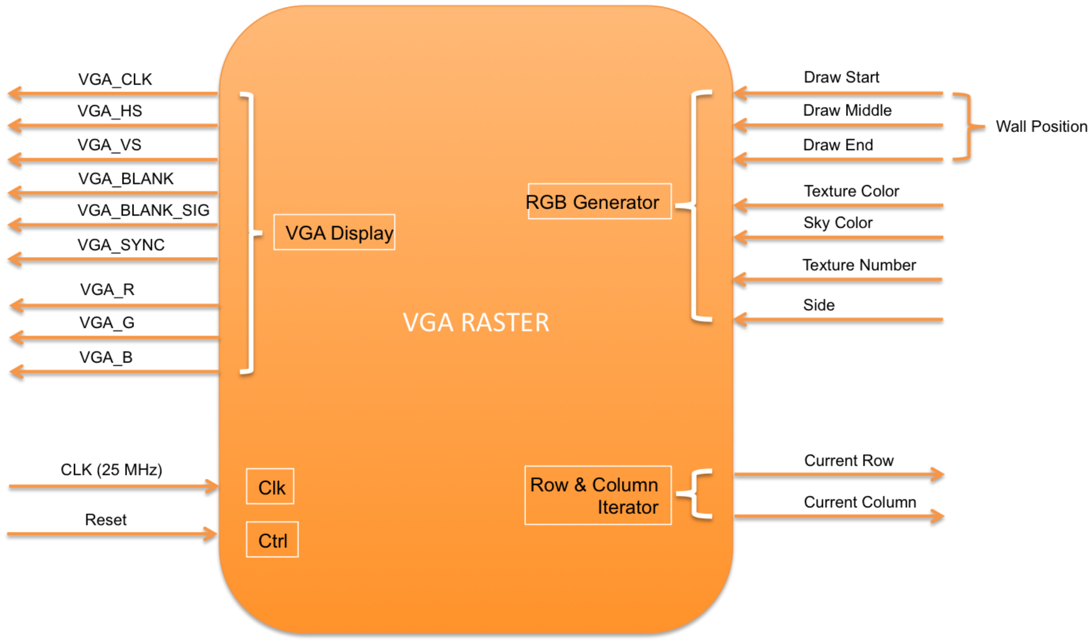


Figure 10. VGA Raster module

VGA Raster is the component that outputs corresponding signals to VGA display to draw colors pixel by pixel. The block interface is shown in Figure 10. It uses a 25 MHz clock to match the one that is used by the VGA display. The component basically consists of the following three parts (1) Row and Column Iterators, (2) Pixel Multiplexer and (3) VGA Timings. Figure 11 shows the detailed structure of VGA raster.

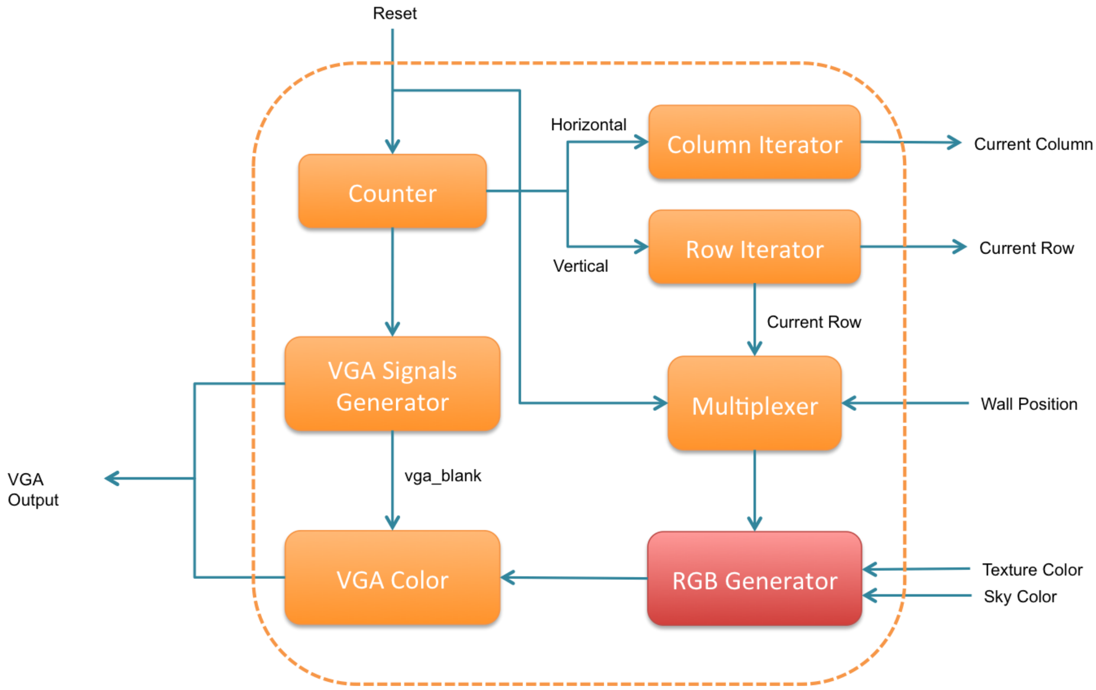


Figure 11. VGA raster internals

Row and Column Iterators. A base counter maintains two values in the horizontal and vertical direction that represent the index of the current row and column currently being output to the display. It will roll back and start again when is has iterated through all pixels or reset signal is set. After accounting for the corresponding VGA constant parameters, the indexes are parsed as iterators for the 640 * 480 display area, which can be used to pull data from memory to calculate texture color information. Further details of how the counter output interacts and controls the following texture calculation will be covered in next section.

Pixel Multiplexer. For a single pixel at given column, the raster will decide to use color data either from texture or sky depending on which range the current row index belongs to. The decision is made from the multiplexer, which generates a logic signal based on the comparison between the current row index and wall start position. Then an asynchronous RGB generator will parse the selected color information into separated RGB codes for monitor.

VGA Timings. The VGA signals, except RGB, are generated from state machines based on horizontal and vertical counters. Also provided, is a reset signal to restart the counter and refresh the screen. Combining the RGB data with other VGA synchronous signals, the final output of VGA raster will be passed to the monitor. Finally, the color output is also controlled by reset and vga_blank signals that can simply refresh parts of the screen to black.

Texture Generation

Figure 12 illustrates the internal structure of texture generation which is included inside the dashed block and its interaction with other modules as well. We give the floor logic a bigger block because it actually takes longer time to finish and is the critical timing path, which will be discussed in the following section.

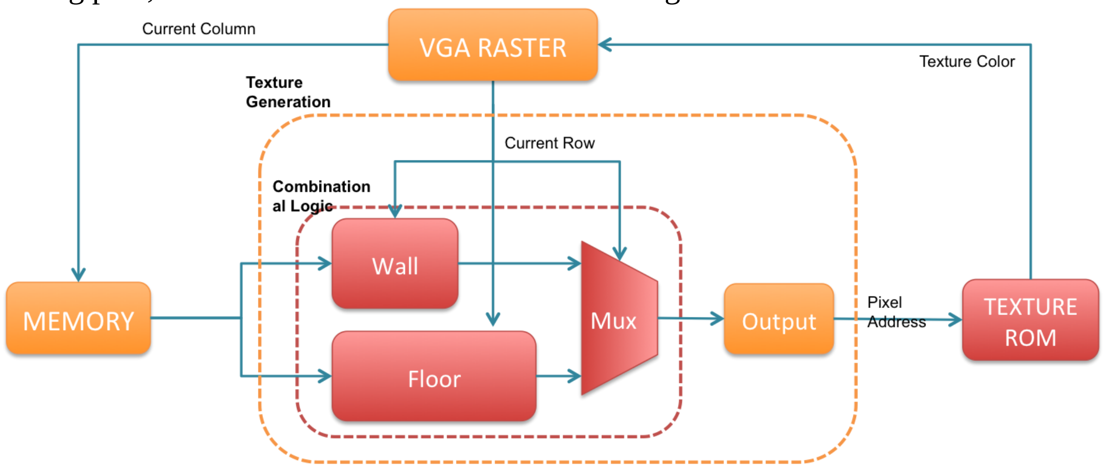


Figure 12. Texture Generation and Floor internals

At a high level, the Texture Generation module maps a 64x64 pixel texture to a wall/floor from the vantage point of the player. It received as an input the index of the pixel on a 640 by 480 screen. Its output is the address of this color data, and the actual data will be fetched from the texture rom, where we pre-loaded four textures with 24 bits of color data for each pixel. Since we have different logic paths to calculate the addresses for the walls and floor, it keeps two combinational logic paths running in parallel. Before outputting the pixel address, it uses a multiplexer to choose whether to draw the wall or floor by checking which range the current row index belongs to. Although the calculation logic is asynchronous, we still want to keep the texture address synchronous in order to match the rate of the VGA raster pipeline structure.

Critical Timing Path

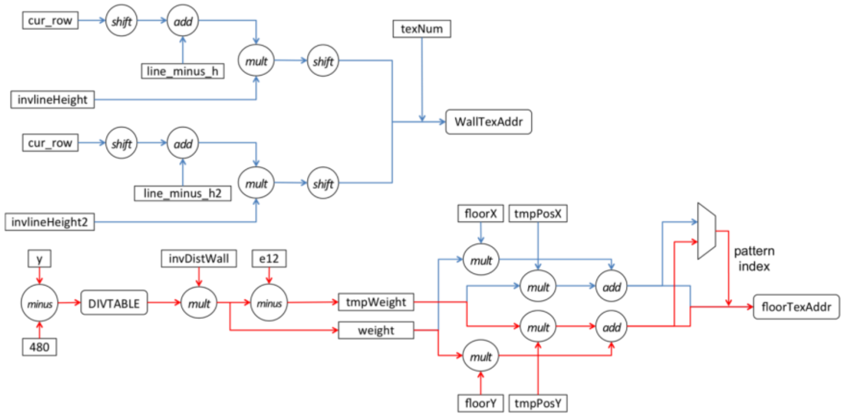


Figure 13. Critical Timing Path

As mentioned above, the texture generation mainly consists of two separate combinational logic paths. The time to calculate wall and floor textures determines the timing limitation of the display. Figure 13 shows the detailed logic diagram of the circuits in texture generation. The highlighted red path in floor logic is our critical timing path. Because of the three multiplications necessary for this step, floor texture generation takes a longer time than generating a wall texture.

Overall VGA Pipeline Structure

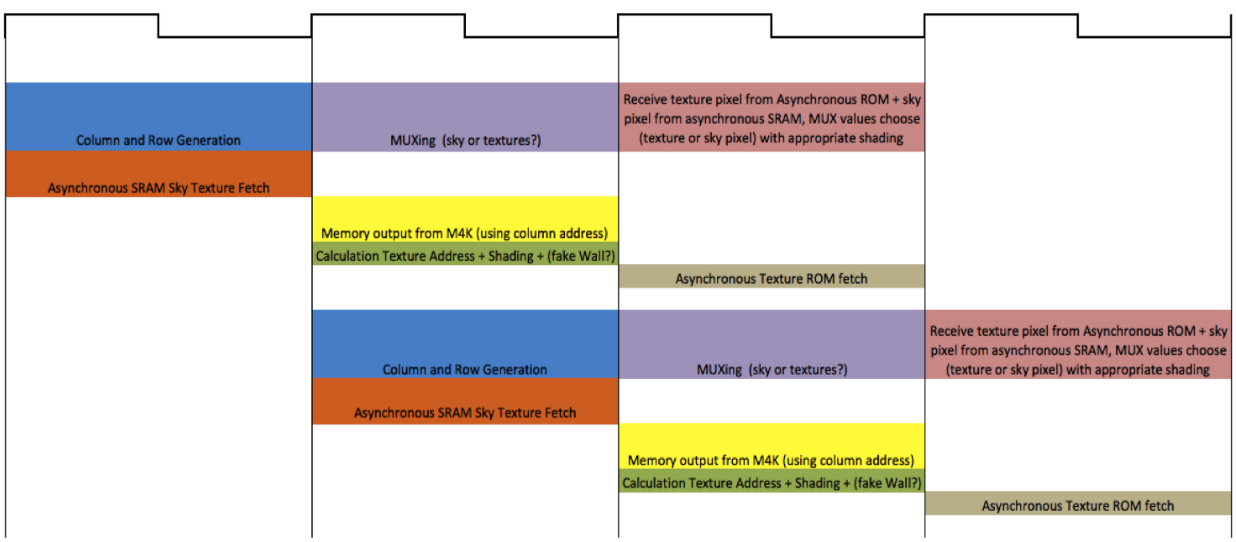


Figure 14. Overall pipeline structure for VGA raster

Figure 14 is the pipeline timing diagram for VGA raster, which illustrates this working flow from a timing perspective. Since the different blocks in VGA raster are performing their own work, and part of the output (column and row) actually decides the value of input (color information from texture and sky) for other blocks, it must guarantee all components are running at the same pace. Therefore, we clocked most of the blocks to form a pipeline structure allowing different blocks to work in parallel at their best efficiency.

The process starts with a new clock cycle by counting the column and row index VGA is going to draw. Then the column number will be passed to memory block to get essential data calculated from the Ray FSM. The row number will be delayed in the same manner, before entering the texture generator. Afterwards, texture addresses for both wall and floor are generated in the asynchronous logic, and the correct one will be picked by the multiplexer. Since we clocked the output, the address is ready at the next stage of the pipeline. Then the color of this pixel can be fetched and immediately calculated from the asynchronous texture rom/SRAM model. Finally, the 24 bit color value is passed to RGB generator block in the VGA Raster module which determines appropriate shading and converts the value to the 30 bit format needed by the monitor.

Sky Generation

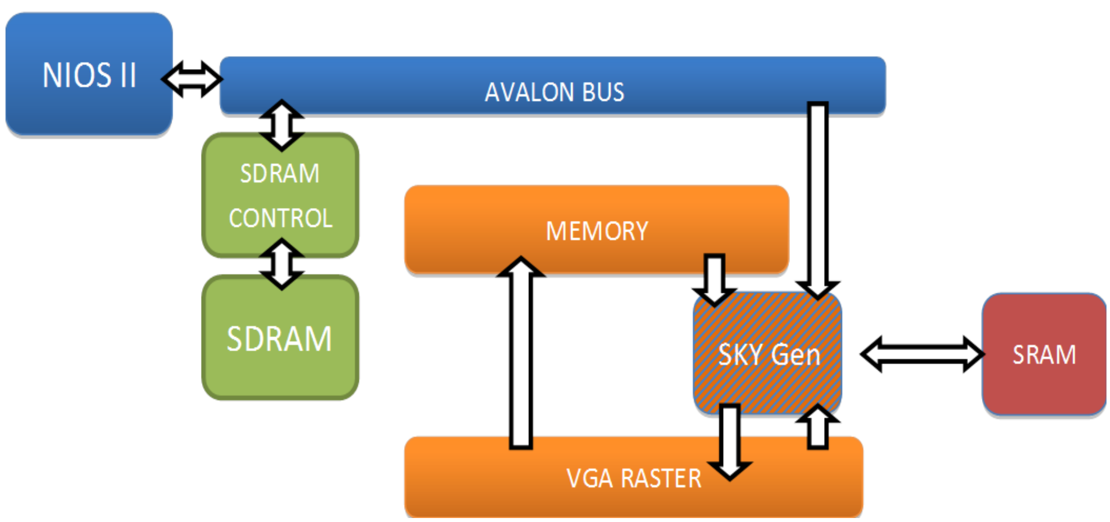


Figure 15. Sky generator system diagram

Sky generator is a module that allows the NIOS to transfer data from SDRAM to SRAM. It also allows the VGA raster module to fetch sky information through it to generate the sky image. With sky generator, we can download various pictures as sky during system initialization. In effect, it replaces the latency to access data from SDRAM for the much faster SRAM.

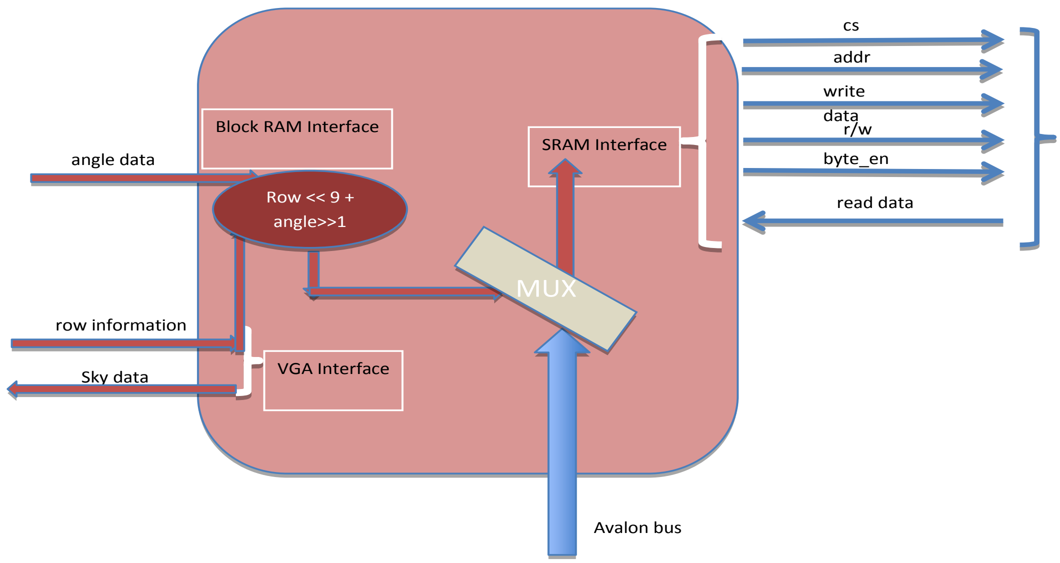


Figure 16. Sky generator design diagram

Figure 16 shows the block diagram of sky generator. The MUX is controlled by NIOS system. When switching to the VGA interface, the sky generator will fetch sky information according to row and angle information. The sky texture is 1024 * 480 * 8b, which is 480KB and enough to fit in the SRAM (512 KB). Since the display range is 640x480, when we rotate to different direction, we will continuously update data from SRAM. SRAM serves as a ring buffer, when runs to the boundary, it will ring back to the beginning column. However, the sky information in SRAM is not enough to provide a full 360 degree view angle. Therefore, we will generate a sky texture in software that will give smooth transitions at boundary transitions.

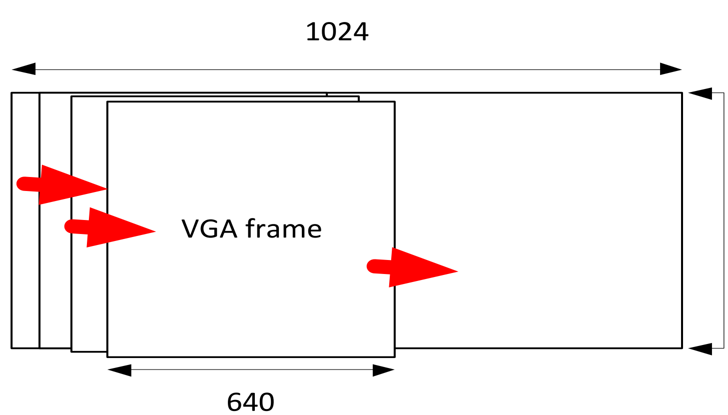


Figure 17. Sky generator access SRAM

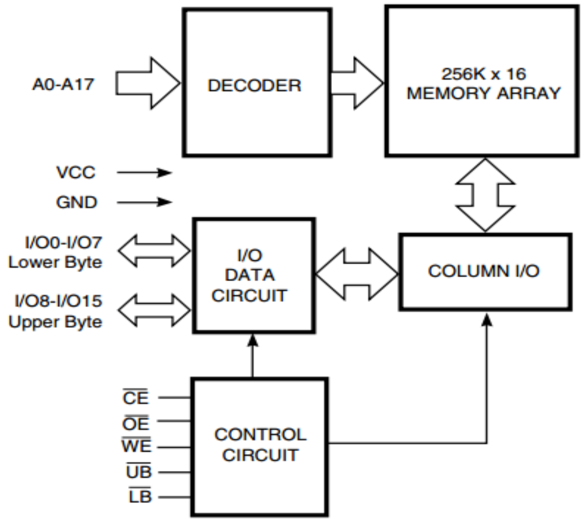


Figure 18. SRAM diagram

It is because the SRAM is asynchronous module, which it dose not have a clock and controlled by input address information, we can set up two different clock domain control interface to access SRAM. The most important one is that maximum read data latency is 15 ns, so the design should be take care of the timing of accessing protocol.

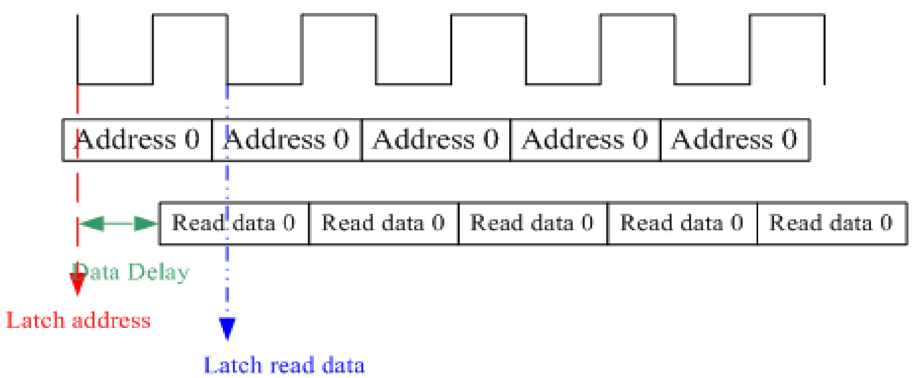


Figure 19. Access SRAM timing

While switching the address, the read data would be valid after a specific latency; the VGA raster must latch data in the next cycle. Since VGA raster only runs at 25MHz clock domain, the timing of the design still get a lot of margin.

SDRAM

NIOS system puts its memory in DRAM and communicates with it with a SDRAM controller. The protocol of DRAM is complicate that the memory controller is not easy to design.

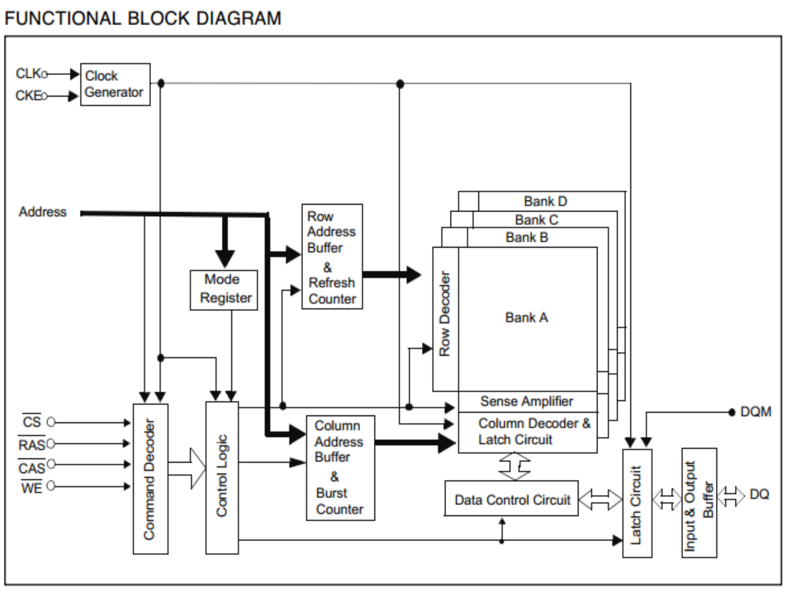


Figure 20. SDRAM diagram

However, the SOPC provide us a well-designed memory controller which also support burst accessing mode that can improve the performance of system. The timing of DRAM interface is also critical, there is a 3 ns timing phase shift from DRAM clock to system clock. Hence, we need to set up a PLL to compensate the timing phase shift for system stability.

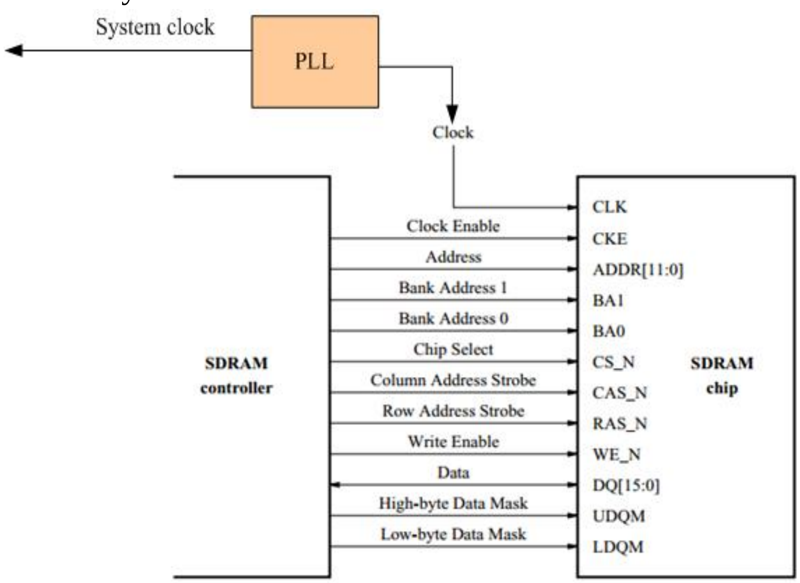


Figure 21. SDRAM interface

Keyboard

We reused the keyboard controller from Lab3. The controller receives data through PS2 serial interface. It was modified so that while receiving a data token from the keyboard, it stores data information to a register. The NIOS system will update the player position after polling from the keyboard.

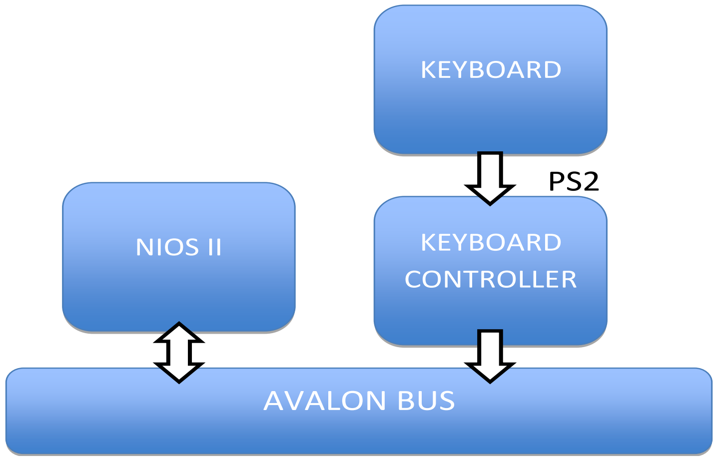


Figure 22. Keyboard sytem diagram

Audio

Figure 23 shows the critical components for playing the background music in our architecture. All the music data has been sampled and programmed in CFI Flash in advance and during the game, NIOS system can fetch new notes from the Flash memory and play them out at the DAC component WM8731. The sound controller works as the interrupt sender in response to the data request from WM8731 and the buffer to hold new data.

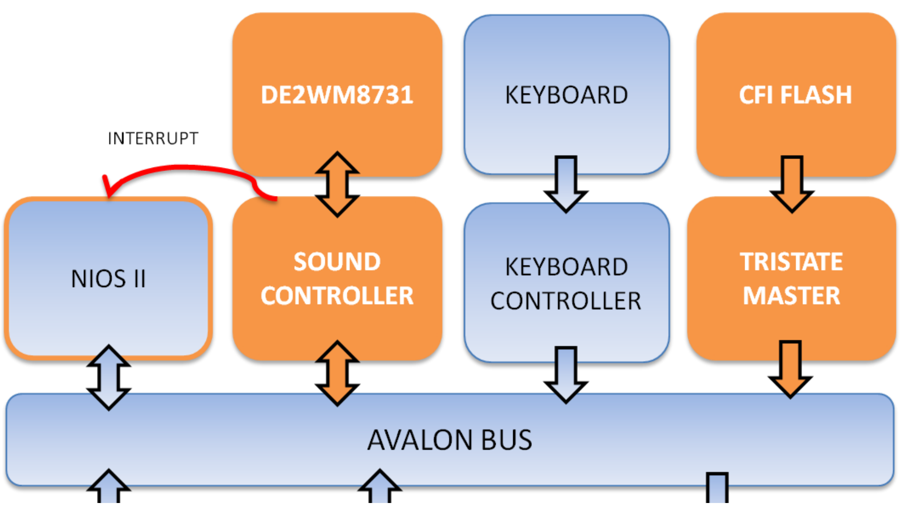


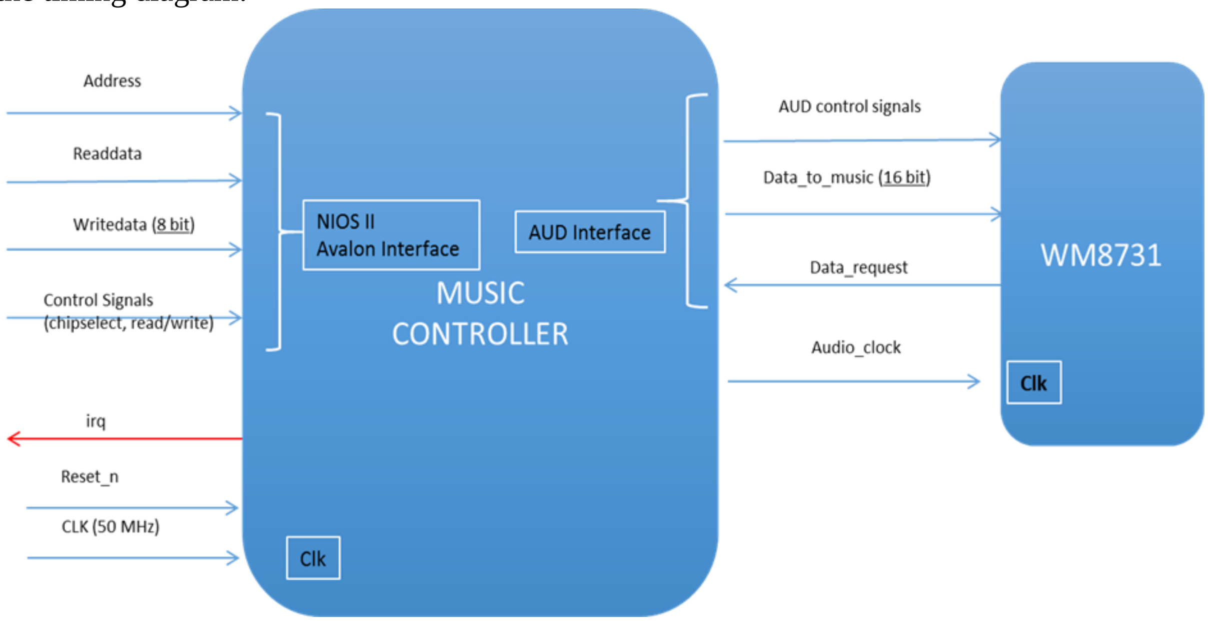
Figure 23. Sound controller interface

Sound Controller

The sound controller plays two roles: one is to get data from the Avalon bus and send the interrupt signal, the other is to buffer the temporary data. Here is the diagram of this part:

Interrupt Sender

To complete data transmission at the interrupts, we first refer to how WM8731 works in the timing diagram:



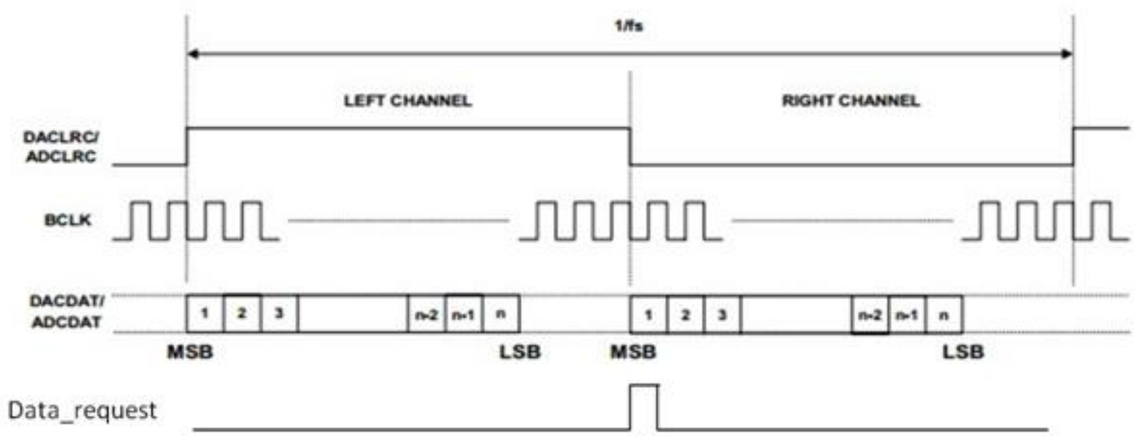


Figure 24 & 25. Audio controller and timing diagram

WM8731 has two channels: the left and the right channel. In our VHDL implementation, the right channel is free so in this period it can fetch data for the next cycle. The data request is sent to tell that a left channel is over and at this time it wants a new note. In a naive implementation, it can be directly connected to the storage interface and receive data in the next location. Here we choose to make use of the Avalon bus, implementing a sound controller with the state machine:

Flash Memory

We considered storing an entire music track or combine several into one track and play it during the game. We tried different storage media for this target: ROM, SDRAM and the Flash. After comparison, the Flash memory became the final choice due to the following factors:

(1) ROM is the simplest non-volatile storage to use and it can be easily configured with the help of MegaWizard. Moreover, hooking it up with WM8731 is easy as exploring the ROM address space. It doesn't need any software but the most important drawback is the limited storage. Through our attempts, ROM has enough capacity to last 6~7 seconds with the sampling rate 22kHz.

(2) SDRAM has 8MB size which is enough in capacity. The data can be loaded as an array written in a header file in NIOS software and programmed automatically into SDRAM when we start to run the program. However in our design, the sky generator takes up some storage in SDRAM so we turn to use the Flash for audio data.

(3) The Flash memory on DE2 board has 4MB volume with 8-bit data width. It is suitable in size since a piece of sound lasting for 90 seconds takes up 440KB in binary file. The Flash also has other advantages that it is easy to transfer, erase and program data to it. The speed is proper since audio does not require a rather high rate. Besides, with the built-in components, we don't need much code to make it work.

Architecture

To use the Flash, the instructions in [3] were followed to build its interface in SOPC builder. We need to add an Avalon-MM Tristate Bridge (under Bridges and Adaptors->Memory Mapped) and a CFI Flash Memory Interface (under Memories and Memory Controllers->Flash). Follow the parameters there and note that the S29GL032N Flash chip on board only supports 8-bit width. Finally connect the conduit signals in the top level.

Data conversion

Audio files are sampled in Matlab. Matlab provides a useful sampling function *wavread()* (in the lastest version the function *audioread()* can support more audio formats including mp3). WAV format requires that the audio be sampled as 8-bit or 16-bit data at a fixed frequency (usually 44kHz). If necessary, the data can be resampled at another frequency with the function *resample()*.

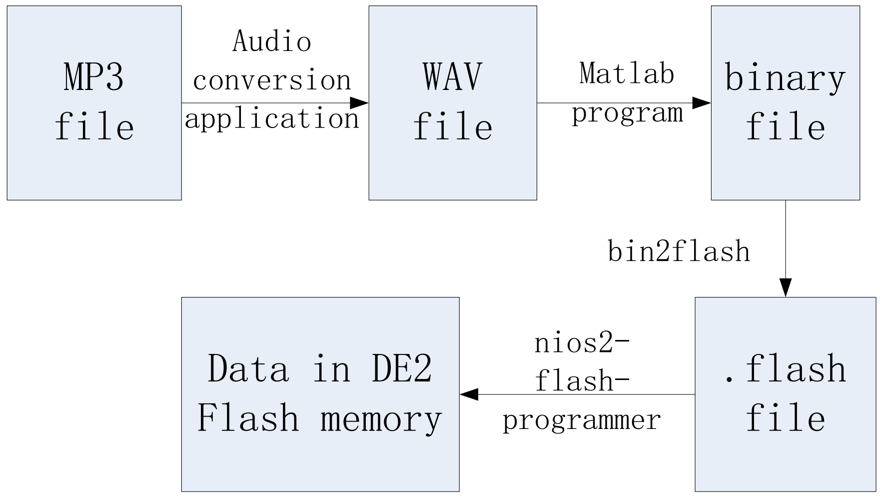


Figure 26. Sound controller interface

Notice the raw results of the *wavread()* function. The original sampling frequency and the data width are pointed out there. The number of samples is also important because it is useful in the software when we want to retrieve data at the certain address range. In addition, since the sampling data is in the range [-1,1], we need a simple conversion to restore them to 8/16 bits.

As long as we obtain a binary file and have configured the Flash to Avalon-bus, the next step is to use the NIOS flash programmer [2] to load data. You can choose to enter the flash-programmer GUI in NIOS2 IDE or use the command prompt:

```
bin2flash --input=sound.bin --output=sound.flash --location=0
nios2-flash-programmer -b 0x400000 --program sound.flash
```

In the above example, the base address of the CFI Flash is assigned as 0x400000.

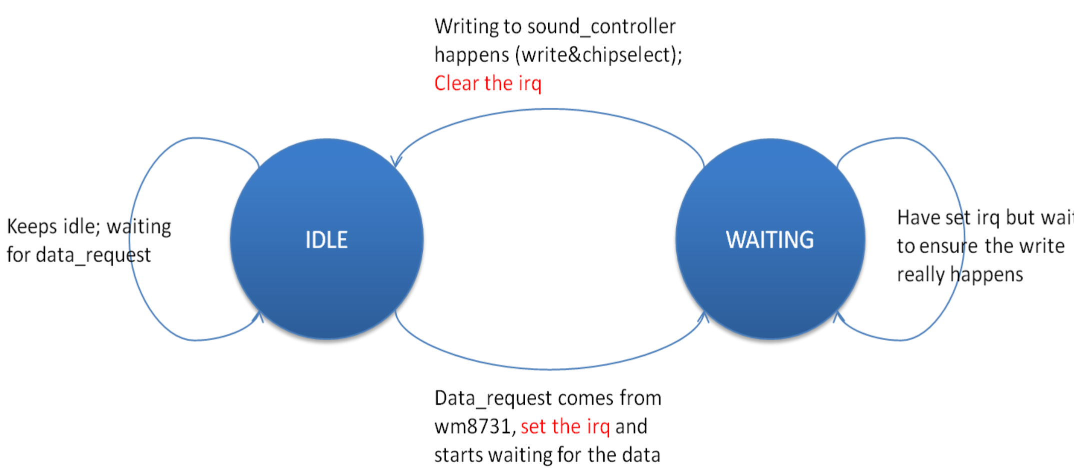


Figure 27. Sound controller interface

The transition between the two states is important as the irq signal is set and cleared. To coordinate it, the interrupt handler *note_isr* in our software reads from CFI Flash and writes to sound controller's buffer. Furthermore, as interrupts stops the main function and makes the overall implementation slower, it is reasonable to establish a buffer with more than one slot that we can reduce the number of interrupts and load more data in one interrupt.

Lessons Learned

Plan/think ahead: Make a software model, and think out the design in hardware before hand. For example, this helped us in foregoing SRAM for frame buffer, allowing us to achieve 60 Hz frame synced frame rate, and also being able to free up the SRAM for the sky.

Stay persistent: There were a few bugs that were beyond reach of TimeQuest Timing Analyzer. Erratic (apparently timing) errors would appear on the screen, such as the line glitch caused by memory buffer switching. However, after thinking and thinking, we were able to get rid of the line using the “patch” memory

Pick a project that is interesting: I enjoyed my hours in the lab, since I enjoyed the project. Working on something you like is much easier, and allows you to do more. For example, many of my ideas came while in the shower or laying in bed. Sounds lame, yes, but it allowed me to contribute more to the group and the project.

Don’t forget hardware debugging. The best way to debug the interrupts is making use of the hardware, for instance, the LEDs. As we have mentioned, printing data on the console caused problems. Thus displaying the information on LEDs or segment displays is a better way.

Use Interrupts wisely. Interrupts can make the main program slower so think twice before using it and try to decrease the rate at which they happen. Moreover, it is wise to disable the interrupts when initializing the program and enable them afterwards. The interrupts can be an obstacle in speed when the main function has a large workload.

Don’t discount timing problems. ModelSim is a good way to check for logic errors, but correct simulation results are just one part of the entire debugging process. Always assuming logic and calculations to be the main reason for bugs ended up being a wrong direction and wasted a lot of time. Learning the timings required by a certain peripherals was the key to getting the project to work properly.

Build and test incrementally. It is important to go forward with your plan step by step and test incrementally. When designing a new module, it may be beneficial to build a local project environment. Even you are confident with the quality of your design, there are many unexpected scenario would happen. Within your own local simulation environment is much easier for you to localize the problem before and after integrating the component.

Don't use printf in interrupts. It is worth mentioning that the *printf* statement cannot be used in the interrupt handler because it occupies the JTAG to communicate with the console and may stop everything. Other unexpected problems tend to happen as well.

Pay attention to clock domains. Use one PLL to generate all clocks in your system. Use a FIFO to transfer between clock domains. Read and know the specifications for your component before you design. All these tips may save you a lot of trouble at the end of the day.

Responsibilities

Alden

Adapting algorithm for hardware
Fixed point software versions
Hardware acceleration
Clock domain organization
Memory module control
VGA debugging

Eddy

High level system diagram
Remodeling state machine
VGA rastering and debugging
Group organization
System interconnection between components

Mingyun

All audio components
Flash programming
Integration of audio with software via interrupts

Weihow

Build system with SDRAM
SRAM control for sky generation
Keyboard integration

Yiming

Combinational logic for hardware acceleration
Floor and texture generation modules

References

[1] Lode Vandevenne, Lode's Computer Graphics Tutorial, 2007 <http://lodev.org/ cgtutor/raycasting.html>.

[2] Brock J. LaMeres, Flash_Programming_the_Altera_DE2_Board, Montana State University, 2013.

[3] Nios II Flash Programmer User Guide, Altera, Feb 2010.

Appendix

de2_ps2.vhd

```
========================================================================
-----------------------------------------------------------------------
-- Simple (receive-only) PS/2 controller for the Altera Avalon bus
-- Presents a two-word interface:
-- Byte 0: LSB is a status bit: 1 = data received, 0 = no new data
-- Byte 4: least significant byte is received data,
--         reading it clears the input register
--
-- Make sure "Slave addressing" in the interfaces tab of SOPC Builder's
-- "New Component" dialog is set to "Register" mode.
-- From an original by Bert Cuzeau
-- (c) ALSE. http://www.alse-fr.com
-----------------------------------------------------------------------
-- Possible improvement : add TIMEOUT on PS2_Clk while shifting
-- Note: PS2_Data is resynchronized though this should not be
-- necessary (qualified by Fall_Clk and does not change at that time).
-- Note the tricks to correctly interpret 'H' as '1' in RTL simulation.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PS2_Ctrl is
  port(
    Clk       : in  std_logic;  -- System Clock
    Reset     : in  std_logic;  -- System Reset
```

```
    PS2_Clk   : in  std_logic;  -- Keyboard Clock Line
    PS2_Data  : in  std_logic;  -- Keyboard Data Line
    DoRead    : in  std_logic;  -- From outside when reading the scan
code
    Scan_Err  : out std_logic;  -- To outside : Parity or Overflow
error
    Scan_DAV  : out std_logic;  -- To outside when a scan code has
arrived
    Scan_Code : out unsigned(7 downto 0) -- Eight bits Data Out
    );
end PS2_Ctrl;

architecture rtl of PS2_Ctrl is

  signal PS2_Datr  : std_logic;

  subtype Filter_t is unsigned(7 downto 0);
  signal Filter    : Filter_t;
  signal Fall_Clk  : std_logic;
  signal Bit_Cnt   : unsigned (3 downto 0);
  signal Parity    : std_logic;
  signal Scan_DAVi : std_logic;

  signal S_Reg     : unsigned(8 downto 0);

  signal PS2_Clk_f : std_logic;

  Type   State_t is (Idle, Shifting);
  signal State : State_t;

begin

  Scan_DAV <= Scan_DAVi;

-- This filters digitally the raw clock signal coming from the keyboard
:
--  * Eight consecutive PS2_Clk=1 makes the filtered_clock go high
--  * Eight consecutive PS2_Clk=0 makes the filtered_clock go low
-- Implies a (FilterSize+1) x Tsys_clock delay on Fall_Clk wrt Data
-- Also in charge of the re-synchronization of PS2_Data

  process (Clk)
  begin
    if rising_edge(Clk) then
      if Reset = '1' then
        PS2_Datr  <= '0';
        PS2_Clk_f <= '0';
        Filter    <= (others => '0');
        Fall_Clk  <= '0';
      else
        PS2_Datr <= PS2_Data and PS2_Data; -- also turns 'H' into '1'
        Fall_Clk <= '0';
        Filter   <= (PS2_Clk and PS2_CLK) & Filter(Filter'high downto
1);
        if Filter = Filter_t'(others=>'1') then
          PS2_Clk_f <= '1';
        elsif Filter = Filter_t'(others=>'0') then
```

```
          PS2_Clk_f <= '0';
          if PS2_Clk_f = '1' then
            Fall_Clk <= '1';
          end if;
        end if;
      end if;
    end if;
  end process;

-- This simple State Machine reads in the Serial Data
-- coming from the PS/2 peripheral.

  process(Clk)
  begin
    if rising_edge(Clk) then
      if Reset = '1' then
        State     <= Idle;
        Bit_Cnt   <= (others => '0');
        S_Reg     <= (others => '0');
        Scan_Code <= (others => '0');
        Parity    <= '0';
        Scan_DAVi <= '0';
        Scan_Err  <= '0';
      else

        if DoRead = '1' then
          Scan_DAVi <= '0'; -- note: this assgnmnt can be overriden
        end if;

        case State is

          when Idle =>
            Parity  <= '0';
            Bit_Cnt <= (others => '0');
            -- note that we do not need to clear the Shift Register
            if Fall_Clk='1' and PS2_Datr='0' then -- Start bit
              Scan_Err <= '0';
              State <= Shifting;
            end if;

          when Shifting =>
            if Bit_Cnt >= 9 then
              if Fall_Clk = '1' then -- Stop Bit
                -- Error is (wrong Parity) or (Stop='0') or Overflow
                Scan_Err  <= (not Parity) or (not PS2_Datr) or
Scan_DAVi;
                Scan_Davi <= '1';
                Scan_Code <= S_Reg(7 downto 0);
                State <= Idle;
              end if;
            elsif Fall_Clk = '1' then
              Bit_Cnt <= Bit_Cnt + 1;
              S_Reg <= PS2_Datr & S_Reg (S_Reg'high downto 1); -- Shift
right
              Parity <= Parity xor PS2_Datr;
            end if;
```

```
          when others => -- never reached
            State <= Idle;

        end case;

        --Scan_Err <= '0'; -- to create a deliberate error

      end if;

    end if;

  end process;

end rtl;

-----------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_ps2 is

  port (
    clk        : in std_logic;
    reset      : in std_logic;

    address    : in std_logic;
    read       : in std_logic;
    chipselect : in std_logic;
    readdata   : out std_logic_vector(7 downto 0);

    PS2_Clk           : in std_logic;
    PS2_Data          : in std_logic
    );
end de2_ps2;

architecture rtl of de2_ps2 is

  signal Data          : unsigned(7 downto 0);
  signal Data_in       : unsigned(7 downto 0);
  signal DataLock      : unsigned(7 downto 0) := "00000000";
  signal DataAvailable : std_logic ;
  signal DataAvailable_in : std_logic;
  signal reg : std_logic := '0';
  signal DoRead        : std_logic;
  type state_type is (A, B, C, D, E);
  signal state   : state_type := A;
  signal inc      : unsigned (7 downto 0) := "11111111";


begin

  U1: entity work.PS2_CTRL port map(
    Clk       => clk,
    Reset     => reset,
    DoRead    => DoRead,
```

```
    PS2_Clk   => PS2_Clk,
    PS2_Data  => PS2_Data,
    Scan_Code => Data_in,
    Scan_DAV  => DataAvailable_in );

  process (clk)
  begin
    if rising_edge(clk) then
      DoRead <= read and chipselect and address;
            DataAvailable <= DataAvailable_in;

            case state is
                        when A=>
                              Data <= Data_in;
                              if Data_in = x"F0" and DataAvailable_in =
'1' then
                                     state <= B;
                              else
                                     state <= A;
                              end if;
                              DataLock <= Data_in;
                              reg <= '0';
                        when B=>
                              Data <= Data_in;
                              if Data_in = x"F0" and DataAvailable_in =
'1' then
                                     state <= B;
                              else
                                    state <= C;
                              end if;
                              DataLock <= Data_in;
                              reg <= '0';
                        when C=>
                              Data <= Data_in - 32;
                              --hold state if value doesn't change
                              if (DataAvailable_in = '1') then
                                    reg <= '1';
                              end if;

                              DataLock <= Data_in;
                              -- if data hasn't become available or
data hasn't changed
                              if (DataLock = Data_in or reg = '0') then
                                    state <= C;
                              else
                                    state <= A;
                              end if;

                        when others =>
                              state <= A;
            end case;

    end if;
  end process;

  process (Data, DataAvailable, address, chipselect)
  begin
```

```
    if chipselect = '1' then
      if address = '1' then
        readdata <= std_logic_vector(Data);
      else
        readdata <= "0000000" & DataAvailable;
      end if;
    else
      readdata <= "00000000";
    end if;
  end process;

end rtl;
========================================================================
```

de2_sram_controller.vhd

```
========================================================================
library ieee;
use ieee.std_logic_1164.all;

entity de2_sram_controller is

  port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address  :  in std_logic_vector(17 downto 0);
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N            : out std_logic
    );

end de2_sram_controller;

architecture dp of de2_sram_controller is
begin

  SRAM_DQ <= writedata when write = '1'
             else (others => 'Z');
  readdata <= SRAM_DQ;
  SRAM_ADDR <= address;
  SRAM_UB_N <= not byteenable(1);
  SRAM_LB_N <= not byteenable(0);
  SRAM_WE_N <= not write;
  SRAM_CE_N <= not chipselect;
  SRAM_OE_N <= not read;

end dp;
========================================================================
```

de2_vga_raster.vhd

```
========================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 25.125 MHz
       bool     : in std_logic;

    VGA_CLK,                         -- Clock
    VGA_HS,                          -- H_SYNC
    VGA_VS,                          -- V_SYNC
    VGA_BLANK,                       -- BLANK
       VGA_BLANK_SIG,
    VGA_SYNC       : out std_logic;      -- SYNC
    VGA_R,                           -- Red[9:0]
    VGA_G,                           -- Green[9:0]
    VGA_B          : out unsigned(9 downto 0); -- Blue[9:0]

       is_Side     : in std_logic;
--  line_height : in unsigned (8 downto 0);
       Col_Color   : in unsigned (23 downto 0);
--     Flr_Color   : in unsigned (7 downto 0);
       Col_Color_sky : in unsigned (7 downto 0);

       Row_Start   : in unsigned (8 downto 0);
       Row_Mid     : in unsigned (8 downto 0);
       Row_End     : in unsigned (8 downto 0);
       texNum      : in unsigned (3 downto 0);
       texNum2      : in unsigned (3 downto 0);

       Cur_Row : out unsigned(9 downto 0);
       Cur_Col : out unsigned(9 downto 0)

    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
```

```
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  constant TEXTURE_HSTART : integer := 0;
  constant TEXTURE_HEND   : integer := 64;
  constant TEXTURE_VSTART : integer := 0;
  constant TEXTURE_VEND   : integer := 64;

  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;

  signal write_pixel :std_logic;
  signal tex_Col : unsigned(5 downto 0);
  signal col_draw_prev : std_logic;
  signal col_draw_sky_prev : std_logic;


  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync : std_logic;  -- Sync. signals

  signal Col_Draw : std_logic;  -- Column Signals area
  signal Col_Draw_sky : std_logic;  -- Column Signals area
  signal R, G, B : unsigned(9 downto 0);
  --signal R_sky, G_sky, B_sky : unsigned(9 downto 0);
  signal ROM_OUT : unsigned(7 downto 0);

  signal     Cur_Row_local : unsigned(9 downto 0);

  signal Texture_h, Texture_v, Texture : std_logic;  -- texture area

  signal Floor_Draw : std_logic;
  --signal Rf, Gf, Bf : unsigned(9 downto 0);

begin

  -- Horizontal and vertical counters

  HCounter : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
  end process HCounter;

  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

  VCounter: process (clk)
  begin
```

```
  if rising_edge(clk) then
    if reset = '1' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' or EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vsync <= '1';
    elsif EndOfLine ='1' then
      if EndOfField = '1' then
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;

VBlankGen : process (clk)
```

```
  begin
    if rising_edge(clk) then
      if reset = '1' then
        vga_vblank <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;

  -- Rectangle generator

  ColumnHGen : process (clk)
  begin
    if rising_edge(clk) then
--      if reset = '1' or Hcount = HSYNC + HBACK_PORCH +
RECTANGLE_HSTART then
--        rectangle_h <= '1';
--      elsif Hcount = HSYNC + HBACK_PORCH + RECTANGLE_HEND then
--        rectangle_h <= '0';
--      end if;
            if Hcount - HSYNC - HBACK_PORCH < 640 then
                  Cur_Col <= Hcount - HSYNC - HBACK_PORCH;
                  tex_Col <= Hcount(5 downto 0) - HSYNC - HBACK_PORCH;
            else
                  Cur_Col <= (others => '0');
                  tex_Col <= (others => '0');
            end if;
    end if;
  end process ColumnHGen;

 CurRowGen : process (clk)
  begin
    if rising_edge(clk) then
            if Vcount - VSYNC - VBACK_PORCH -1 < 480 then
                  Cur_Row <= Vcount - VSYNC - VBACK_PORCH - 1;
                  Cur_Row_local <= Vcount - VSYNC - VBACK_PORCH - 1;
            else
                  Cur_Row <= (others => '0');
                  Cur_Row_local <= (others => '0');
            end if;
    end if;
  end process CurRowGen;

  ColumnVGen : process (clk)
  begin
    if rising_edge(clk) then
            --col_draw_prev <= col_draw;
      if reset = '1' then
                  Col_Draw <= '0';
            elsif (Cur_Row_local > Row_Start or Cur_Row_local >
Row_Mid) then
                  Col_Draw <= '1';
```

```
                  if (Cur_Row_local >= Row_End) then
                        Floor_Draw <= '1';
                  else
                        Floor_Draw <= '0';
                  end if;
            else
                  Col_Draw <= '0';
      end if;
    end if;
  end process ColumnVGen;


  ColumnVGen_sky : process (clk)
  begin
    if rising_edge(clk) then
            --col_draw_sky_prev <= col_draw_sky;
      if reset = '1' then
                  Col_Draw_sky <= '0';
            elsif (Cur_Row_local <= Row_Start and Cur_Row_local <=
Row_Mid) then
                  Col_Draw_sky <= '1';
            else
                  Col_Draw_sky <= '0';
      end if;
    end if;
  end process ColumnVGen_sky;

--   FloorVGen : process (clk)
--  begin
--    if rising_edge(clk) then
--      if reset = '1' then
--                Floor_Draw <= '0';
--          elsif (Cur_Row_local >= Row_End) then
--                Floor_Draw <= '1';
--          else
--                Floor_Draw <= '0';
--          end if;
--    end if;
--  end process FloorVGen;

  ColorGen : process(Col_Color,Col_Color_sky,col_draw_sky, col_draw,
is_Side, bool, texNum, texNum2)
  variable R_temp : unsigned (9 downto 0);
  variable G_temp : unsigned (9 downto 0);
  variable B_temp : unsigned (9 downto 0);

  variable R_sky : unsigned (9 downto 0);
  variable G_sky : unsigned (9 downto 0);
  variable B_sky : unsigned (9 downto 0);

  begin
--            R_temp := Col_Color(7 downto 5) & Col_Color(7 downto 5) &
Col_Color(7 downto 5) & Col_Color(7);
--        G_temp := Col_Color(4 downto 2) & Col_Color(4 downto 2) &
Col_Color(4 downto 2) & Col_Color(4);
--        B_temp := Col_Color(1 downto 0) & Col_Color(1 downto 0) &
Col_Color(1 downto 0) & Col_Color(1 downto 0) & Col_Color(1 downto 0);
```

```
--
--            R_sky :=  Col_Color_sky(7 downto 5) & Col_Color_sky(7
downto 5) & Col_Color_sky(7 downto 5) & Col_Color_sky(7);
--        G_sky :=  Col_Color_sky(4 downto 2) & Col_Color_sky(4 downto
2) & Col_Color_sky(4 downto 2) & Col_Color_sky(4);
--        B_sky :=  Col_Color_sky(1 downto 0) & Col_Color_sky(1 downto
0) & Col_Color_sky(1 downto 0) & Col_Color_sky(1 downto 0) &
Col_Color_sky(1 downto 0);

             R_sky :=  Col_Color_sky(7 downto 0) & "00";
       G_sky :=  Col_Color_sky(7 downto 0) & "00";
             B_sky :=   Col_Color_sky(7 downto 0) & "00" ;

              R_temp := Col_Color(23 downto 16) & "00" ;
        G_temp := Col_Color(15 downto 8) & "00";
        B_temp := Col_Color(7 downto 0) & "00";

            if (Col_Draw_sky = '1' or (bool = '1' and texNum = x"4") or
(bool = '0' and texNum2 = x"4") ) then
                  R <= R_sky;
                  G <= G_sky;
                  B <= B_sky;

            elsif (col_draw = '1') then
                  if (is_Side = '0') then
                        R <= R_temp;
                        G <= G_temp;
                        B <= B_temp;
                  else
                        R <= (R_temp srl 1);
                        G <= (G_temp srl 1);
                        B <= (B_temp srl 1);
                  end if;
            else
                  R <= "0000000000";
                  G <= "0000000000";
                  B <= "0000000000";
            end if;

  end process ColorGen;

      BlankGen : process(vga_hblank,vga_vblank)
      begin
            if (vga_hblank = '0' and vga_vblank = '0') then
                  write_pixel <= '1';
            else
                  write_pixel <= '0';
            end if;

      end process BlankGen;

--  ColorGen_sky : process(Col_Color_sky)
--  begin
--            R_sky <=  Col_Color_sky(7 downto 5) & Col_Color_sky(7
downto 5) & Col_Color_sky(7 downto 5) & Col_Color_sky(7);
--        G_sky <=  Col_Color_sky(4 downto 2) & Col_Color_sky(4 downto
2) & Col_Color_sky(4 downto 2) & Col_Color_sky(4);
```

```
--        B_sky <=  Col_Color_sky(1 downto 0) & Col_Color_sky(1 downto
0) & Col_Color_sky(1 downto 0) & Col_Color_sky(1 downto 0) &
Col_Color_sky(1 downto 0);
--    end process ColorGen_sky;
--
--    FloorGen : process(Flr_Color)
--  begin
--            Rf <= Flr_Color(7 downto 5) & Flr_Color(7 downto 5) &
Flr_Color(7 downto 5) & Flr_Color(7);
--        Gf <= Flr_Color(4 downto 2) & Flr_Color(4 downto 2) &
Flr_Color(4 downto 2) & Flr_Color(4);
--        Bf <= Flr_Color(1 downto 0) & Flr_Color(1 downto 0) &
Flr_Color(1 downto 0) & Flr_Color(1 downto 0) & Flr_Color(1 downto 0);
--    end process FloorGen;

  -- Registered video signals going to the video DAC

  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
            if write_pixel = '1' then
                  VGA_R <= R;
                  VGA_G <= G;
                  VGA_B <= B;
            else
                  VGA_R <= "0000000000";
                  VGA_G <= "0000000000";
                  VGA_B <= "0000000000";
            end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);
  VGA_BLANK_SIG <= vga_vblank;
end rtl;
========================================================================
```

de2_wm8731_audio.vhd

```
========================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_wm8731_audio is
port (
```

```
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43
MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test
mode
    audio_request : out std_logic;  --    Audio controller request new
data
    data : in unsigned(15 downto 0);

    -- Audio interface signals
    AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
  );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

    signal lrck : std_logic;
    signal bclk : std_logic;
    signal xck  : std_logic;

    signal lrck_divider : unsigned(11 downto 0);
    signal bclk_divider : unsigned(7 downto 0);

    signal set_bclk : std_logic;
    signal set_lrck : std_logic;
    signal clr_bclk : std_logic;
    signal lrck_lat : std_logic;

    signal shift_out : unsigned(15 downto 0);

    signal sin_out     : unsigned(15 downto 0);
    signal sin_counter : unsigned(5 downto 0);

begin

    -- LRCK divider
    -- Audio chip main clock is 18.432MHz / Sample rate 48KHz
    -- Divider is 18.432 MHz / 48KHz = 192 (X"C0")
    -- Left justify mode set by I2C controller

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck_divider <= (others => '0');
      elsif lrck_divider = X"1A1"  then        -- "C0" minus 1
        lrck_divider <= X"000";
      else
        lrck_divider <= lrck_divider + 1;
      end if;
    end if;
  end process;
```

```
process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      bclk_divider <= (others => '0');
    elsif bclk_divider = X"1A" or set_lrck = '1'  then
      bclk_divider <= X"00";
    else
      bclk_divider <= bclk_divider + 1;
    end if;
  end if;
end process;

set_lrck <= '1' when lrck_divider = X"1A1" else '0';

process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      lrck <= '0';
    elsif set_lrck = '1' then
      lrck <= not lrck;
    end if;
  end if;
end process;

-- BCLK divider
set_bclk <= '1' when bclk_divider(7 downto 0) = "00001100" else '0';
clr_bclk <= '1' when bclk_divider(7 downto 0) = "00011001" else '0';

process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      bclk <= '0';
    elsif set_lrck = '1' or clr_bclk = '1' then
      bclk <= '0';
    elsif set_bclk = '1' then
      bclk <= '1';
    end if;
  end if;
end process;

-- Audio data shift output
process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      shift_out <= (others => '0');
    elsif set_lrck = '1' then
      if test_mode = '1' then
        shift_out <= sin_out;
      else
        shift_out <= data;
      end if;
    elsif clr_bclk = '1' then
      shift_out <= shift_out (14 downto 0) & '0';
```

```
    end if;
  end if;
end process;

  -- Audio outputs

  AUD_ADCLRCK  <= lrck;
  AUD_DACLRCK  <= lrck;
  AUD_DACDAT   <= shift_out(15);
  AUD_BCLK     <= bclk;

  -- Self test with Sin wave

  process(clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
          sin_counter <= (others => '0');
      elsif lrck_lat = '1' and lrck = '0'  then
        if sin_counter = "101111" then
          sin_counter <= "000000";
        else
          sin_counter <= sin_counter + 1;
        end if;
      end if;
    end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      lrck_lat <= lrck;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if lrck_lat = '1' and lrck = '0' then
        audio_request <= '1';
      else
        audio_request <= '0';
      end if;
    end if;
  end process;

with sin_counter select sin_out <=
  X"0000" when "000000",
  X"10b4" when "000001",
  X"2120" when "000010",
  X"30fb" when "000011",
  X"3fff" when "000100",
  X"4deb" when "000101",
  X"5a81" when "000110",
  X"658b" when "000111",
  X"6ed9" when "001000",
  X"7640" when "001001",
```

```
    X"7ba2" when "001010",
    X"7ee6" when "001011",
    X"7fff" when "001100",
    X"7ee6" when "001101",
    X"7ba2" when "001110",
    X"7640" when "001111",
    X"6ed9" when "010000",
    X"658b" when "010001",
    X"5a81" when "010010",
    X"4deb" when "010011",
    X"3fff" when "010100",
    X"30fb" when "010101",
    X"2120" when "010110",
    X"10b4" when "010111",
    X"0000" when "011000",
    X"ef4b" when "011001",
    X"dee0" when "011010",
    X"cf05" when "011011",
    X"c001" when "011100",
    X"b215" when "011101",
    X"a57e" when "011110",
    X"9a74" when "011111",
    X"9127" when "100000",
    X"89bf" when "100001",
    X"845d" when "100010",
    X"8119" when "100011",
    X"8000" when "100100",
    X"8119" when "100101",
    X"845d" when "100110",
    X"89bf" when "100111",
    X"9127" when "101000",
    X"9a74" when "101001",
    X"a57e" when "101010",
    X"b215" when "101011",
    X"c000" when "101100",
    X"cf05" when "101101",
    X"dee0" when "101110",
    X"ef4b" when "101111",
    X"0000" when others;

end architecture;
```

==

floorMod.vhd

==

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity floorMod is
      port (clk               : in std_logic;
                  floorX            : in unsigned (17 downto 0);
                  floorY            : in unsigned (17 downto 0);
```

```
                  tmpPosX          : in unsigned (17 downto 0);
                  tmpPosY          : in unsigned (17 downto 0);
                  invDistWall : in unsigned (11 downto 0);
                  y                      : in unsigned (8 downto 0);
                  textureIndexOut:  out unsigned (11 downto 0)
);
end floorMod;

architecture imp of floorMod is

signal y_local : unsigned (8 downto 0);

type rom_type is array(0 to 239) of unsigned (15 downto 0);
constant DIVTABLE: rom_type := (
      x"1000",x"1011",x"1022",x"1033",x"1045",x"1057",x"1069",x"107b",x
"108d",x"109f",x"10b2",x"10c4",x"10d7",x"10ea",x"10fd",x"1111",x"1124",
x"1138",x"114c",x"1160",x"1174",x"1188",x"119d",x"11b2",x"11c7",x"11dc"
,x"11f1",x"1207",x"121c",x"1232",x"1249",x"125f",x"1276",x"128c",x"12a4
",x"12bb",x"12d2",x"12ea",x"1302",x"131a",x"1333",x"134b",x"1364",x"137
e",x"1397",x"13b1",x"13cb",x"13e5",x"1400",x"141a",x"1435",x"1451",x"14
6c",x"1488",x"14a5",x"14c1",x"14de",x"14fb",x"1519",x"1537",x"1555",x"1
573",x"1592",x"15b1",x"15d1",x"15f1",x"1611",x"1632",x"1653",x"1674",x"
1696",x"16b8",x"16db",x"16fe",x"1721",x"1745",x"176a",x"178e",x"17b4",x
"17d9",x"1800",x"1826",x"184d",x"1875",x"189d",x"18c6",x"18ef",x"1919",
x"1943",x"196e",x"1999",x"19c5",x"19f2",x"1a1f",x"1a4d",x"1a7b",x"1aaa"
,x"1ada",x"1b0a",x"1b3b",x"1b6d",x"1ba0",x"1bd3",x"1c07",x"1c3c",x"1c71
",x"1ca8",x"1cdf",x"1d17",x"1d50",x"1d89",x"1dc4",x"1e00",x"1e3c",x"1e7
9",x"1eb8",x"1ef7",x"1f38",x"1f79",x"1fbc",x"2000",x"2044",x"208a",x"20
d2",x"211a",x"2164",x"21af",x"21fb",x"2249",x"2298",x"22e8",x"233a",x"2
38e",x"23e3",x"2439",x"2492",x"24ec",x"2548",x"25a5",x"2605",x"2666",x"
26c9",x"272f",x"2796",x"2800",x"286b",x"28d9",x"294a",x"29bd",x"2a32",x
"2aaa",x"2b25",x"2ba2",x"2c23",x"2ca6",x"2d2d",x"2db6",x"2e43",x"2ed4",
x"2f68",x"3000",x"309b",x"313b",x"31de",x"3286",x"3333",x"33e4",x"349a"
,x"3555",x"3615",x"36db",x"37a6",x"3878",x"3950",x"3a2e",x"3b13",x"3c00
",x"3cf3",x"3def",x"3ef3",x"4000",x"4115",x"4234",x"435e",x"4492",x"45d
1",x"471c",x"4873",x"49d8",x"4b4b",x"4ccc",x"4e5e",x"5000",x"51b3",x"53
7a",x"5555",x"5745",x"594d",x"5b6d",x"5da8",x"6000",x"6276",x"650d",x"6
7c8",x"6aaa",x"6db6",x"70f0",x"745d",x"7800",x"7bde",x"8000",x"8469",x"
8924",x"8e38",x"93b1",x"9999",x"a000",x"a6f4",x"ae8b",x"b6db",x"c000",x
"ca1a",x"d555",x"e1e1",x"f000",x"0000",x"1249",x"2762",x"4000",x"5d17",
x"8000",x"aaaa",x"e000",x"2492",x"8000",x"0000",x"c000",x"0000",x"8000"
,x"0000"
);

begin

      process (floorX, floorY, tmpPosX, tmpPosY, invDistWall, y_local)

      variable currentDist : unsigned (15 downto 0);
      variable weight:  unsigned (27 downto 0);
      variable tmp : unsigned (12 downto 0);
      variable currentFloorX : unsigned (30 downto 0);
      variable currentFloorY : unsigned (30 downto 0);

      variable floorTexX : unsigned (5 downto 0);
      variable floorTexY : unsigned (5 downto 0);
```

```
      begin

            currentDist := DIVTABLE(to_integer(480 - y_local));
            weight := currentDist * invDistWall;
            tmp := "1000000000000" - weight(23 downto 12);

            currentFloorX := (weight(23 downto 12) * floorX + tmp *
tmpPosX);
            currentFloorY := (weight(23 downto 12) * floorY + tmp *
tmpPosY);


            floorTexX := currentFloorX(23 downto 18);
            floorTexY := currentFloorY(23 downto 18);

--          textureIndexOut <= x"000";
            textureIndexOut <= (floorTexY & floorTexX);

      end process;

      process(clk)
      begin
            if (rising_edge(clk)) then
                  y_local <= y;
            end if;
      end process;

end imp;
========================================================================
```

framerate_calc.vhd

```
========================================================================
-----------------------------------------------------------------------
-- Frame Rate Calculation
--
-- Edward Garcia
-- ewg2115@columbia.edu
-----------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity framerate_calc is

  port (
     clk           : in std_logic;                    -- Should be 50
MHz
       wr_addr     : in unsigned (9 downto 0);
       frame_rate : out unsigned (7 downto 0)
    );

end framerate_calc;
```

```
architecture rtl of framerate_calc is

   constant CLOCK_1_SECOND       : integer := 50000000; -- 50MHz
      constant MAX_COLUMN                 : integer := 640;

  -- Signals for the video controller
   signal frame_count : unsigned(7 downto 0) := "00000000";  --
Horizontal position (0-800)
   signal clk_count   : unsigned(25 downto 0) := (others => '0');  --
Vertical position (0-524)
      signal prev_wr_addr : unsigned(9 downto 0) := "0000000000";
      type states is (A, B, B2, C);
      signal state : states := A;

begin

  frame_counter : process (clk, wr_addr)
  begin
  if rising_edge(clk) then
            clk_count <= clk_count + 1;
            prev_wr_addr <= wr_addr;

            case state is
                  when A =>
                        if clk_count = CLOCK_1_SECOND then
                              state <= B;
                        elsif wr_addr = MAX_COLUMN -1 then
                              state <= C;
                        else
                              state <= A;
                        end if;

                  when B =>
                        frame_rate <= frame_count;
                        clk_count <= (others => '0');
                        state <= B2;

                  when B2 =>
                        frame_count <= (others => '0');

                        state <= A;

                  when C =>
                        if wr_addr = MAX_COLUMN -1 then
                              state <= C;
                        else
                              frame_count <= frame_count + 1;
                              state <= A;
                        end if;

                  when others =>
                        state <= A;

                  end case;
            end if;
  end process frame_counter;
```

```
end rtl;
==========================================================================
```

memcustom.vhd

```
==========================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memcustom is
      port
      (
            clock       : in std_logic  := '1';
            data        : in std_logic_vector (255 downto 0);
            rdaddress         : in unsigned(9 downto 0);
            --wraddress : in unsigned(9 downto 0);
            --wren            : in std_logic  := '0';
            rd_req : in std_logic := '0' ;
            --VGA_BLANK : in std_logic := '0';
            row_in  : in unsigned (9 downto 0);
            row_out : out unsigned (9 downto 0);

            q           : out std_logic_vector(255 downto 0)
      );
end memcustom;


architecture rtl of memcustom is

signal toggle : std_logic := '1';
signal blank_prev : std_logic := '0';
signal wren : std_logic  := '0';

signal wraddress : unsigned(9 downto 0);
signal address_1_a : std_logic_vector (8 downto 0);
signal address_1_b : std_logic_vector (6 downto 0);
signal address_2_a : std_logic_vector (8 downto 0);
signal address_2_b : std_logic_vector (6 downto 0);
signal address_patch_1 : std_logic_vector (5 downto 0);
signal address_patch_2 : std_logic_vector (5 downto 0);

signal data_1_a : std_logic_vector (255 downto 0);
signal data_1_b : std_logic_vector (255 downto 0);
signal data_2_a : std_logic_vector (255 downto 0);
signal data_2_b : std_logic_vector (255 downto 0);
signal data_patch_1 : std_logic_vector (255 downto 0);
signal data_patch_2 : std_logic_vector (255 downto 0);

signal q_1_a : std_logic_vector (255 downto 0);
signal q_1_b : std_logic_vector (255 downto 0);
signal q_2_a : std_logic_vector (255 downto 0);
signal q_2_b : std_logic_vector (255 downto 0);
signal q_patch_1 : std_logic_vector (255 downto 0);
signal q_patch_2 : std_logic_vector (255 downto 0);
```

```
signal wren_1_a : std_logic;
signal wren_1_b : std_logic;
signal wren_2_a : std_logic;
signal wren_2_b : std_logic;
signal wren_patch_1 : std_logic;
signal wren_patch_2 : std_logic;

type read_codes is (A, B, C, D, E , F);
signal read_code : read_codes;

begin

   M0: entity work.mem_custom_2_a port map (
            address     => address_1_a,
            clock       => clock,
            data   => data_1_a,
            wren  =>  wren_1_a,
            q     =>    q_1_a
  );

   M1: entity work.mem_custom_2_b port map (
            address     => address_1_b,
            clock       => clock,
            data   => data_1_b,
            wren  =>  wren_1_b,
            q     =>    q_1_b
  );

   M3: entity work.mem_custom_2_a port map (
            address     => address_2_a,
            clock       => clock,
            data   => data_2_a,
            wren  =>  wren_2_a,
            q     =>    q_2_a
  );

   M4: entity work.mem_custom_2_b port map (

            address           => address_2_b,
            clock       => clock,
            data        => data_2_b,
            wren        => wren_2_b,
            q           =>    q_2_b

      );

      M5: entity work.mem_patch_2 port map(

            address           => address_patch_1,
            clock       => clock,
            data        => data_patch_1,
            wren        => wren_patch_1,
            q           =>    q_patch_1

  );
```

```
      M6: entity work.mem_patch_2 port map(
            address           => address_patch_2,
            clock       => clock,
            data => data_patch_2,
            wren => wren_patch_2,
            q           =>    q_patch_2
  );

process (clock)

begin

if (rising_edge(clock)) then

       --data(242) is VGA_BLANK from FSM through FIFO
      --blank_prev <= data(242);
      if wren = '1' and data(242) = '1' then
            toggle <= not toggle;
      end if;

      --read_code
            --00 = 1_a
            --01 = 1_b
            --10 = 2_a
            --11 = 2_b
--    case read_code is
--          when A =>   q <= q_1_a;
--          when B =>   q <= q_1_b;
--          when C =>   q <= q_2_a;
--          when D =>   q <= q_2_b;
--    when E =>   q <= q_patch_1;
--    when F =>   q <= q_patch_2;
--          when others => q <= (others => '0');
--    end case;

      row_out <= row_in;

      if (rd_req = '1') then
            wren <= '1';
      else
            wren <= '0';
      end if;

      if (toggle = '1') then
            if (rdaddress < "0111110000") then
                  read_code <= C;
            elsif (rdaddress >= "1000010000") then
                  read_code <= D;
            else
                  read_code <= F;
            end if;
      else
            if (rdaddress < "0111110000") then
                  read_code <= A;
            elsif (rdaddress >= "1000010000") then
                  read_code <= B;
            else
```

```
                read_code <= E;
            end if;
        end if;
end if;

end process;

-- MUX for address inputs and writes
--
process (data)
begin
        wraddress <= unsigned(data(241 downto 232));
end process;

process(toggle, wren, wraddress,rdaddress,data)

begin

        data_1_a <= data;
        data_1_b <= data;
        data_2_a <= data;
        data_2_b <= data;
        data_patch_1 <= data;
        data_patch_2 <= data;

        if (toggle = '1') then

            if (wren = '1') then
                if (wraddress < "0111111110") then
                    wren_1_a <= '1';
                    wren_1_b <= '0';
                elsif (wraddress >= "1000000010") then
                    wren_1_a <= '0';
                    wren_1_b <= '1';
                else
                    wren_1_a <= '0';
                    wren_1_b <= '0';
                end if;

                if (wraddress >= "0111100010" and wraddress <
"1000011110" ) then
                    wren_patch_1 <= '1';
                else
                    wren_patch_1 <= '0';
                end if;

            else
                wren_1_a <= '0';
                wren_1_b <= '0';
                wren_patch_1 <= '0';
            end if;

            wren_2_a <= '0';
            wren_2_b <= '0';
            wren_patch_2 <= '0';
```

```
            address_1_a <= std_logic_vector (wraddress(8 downto 0));
            address_1_b <= std_logic_vector (wraddress(6 downto 0));
            address_patch_1 <= std_logic_vector (wraddress(5 downto
0));

            address_2_a <= std_logic_vector(rdaddress(8 downto 0));
            address_2_b <= std_logic_vector (rdaddress(6 downto 0));
            address_patch_2 <= std_logic_vector (rdaddress(5 downto
0));

      else

            if (wren = '1') then
                  if (wraddress < "0111111110") then
                        wren_2_a <= '1';
                        wren_2_b <= '0';
                  elsif (wraddress >= "1000000010") then
                        wren_2_a <= '0';
                        wren_2_b <= '1';
                  else
                        wren_2_a <= '0';
                        wren_2_b <= '0';
                  end if;

                  if (wraddress >= "0111100010" and wraddress <
"1000011110" ) then
                        wren_patch_2 <= '1';
                  else
                        wren_patch_2 <= '0';
                  end if;

            else
                  wren_2_a <= '0';
                  wren_2_b <= '0';
                  wren_patch_2 <= '0';
            end if;

            wren_1_a <= '0';
            wren_1_b <= '0';
            wren_patch_1 <= '0';

            address_2_a <= std_logic_vector(wraddress(8 downto 0));
            address_2_b <= std_logic_vector(wraddress(6 downto 0));
            address_patch_2 <= std_logic_vector(wraddress(5 downto 0));

            address_1_a <= std_logic_vector(rdaddress(8 downto 0));
            address_1_b <= std_logic_vector(rdaddress(6 downto 0));
            address_patch_1 <= std_logic_vector(rdaddress(5 downto 0));
      end if;
end process;


-- MUX for output read

process (read_code,q_1_a,q_1_b,q_2_a,q_2_b,q_patch_1,q_patch_2)

begin
```

```
      case read_code is
            when A =>   q <= q_1_a;
            when B =>   q <= q_1_b;
            when C =>   q <= q_2_a;
            when D =>   q <= q_2_b;
            when E =>   q <= q_patch_1;
            when F =>   q <= q_patch_2;
            when others => q <= (others => '0');
      end case;

end process;

end rtl;
========================================================================
```

niosInterface.vhd

```
========================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity niosInterface is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(4 downto 0);

    readdata   : out std_logic_vector(31 downto 0);
    writedata  : in  std_logic_vector(31 downto 0);

    hardware_data : in std_logic_vector(31 downto 0);
    ctrl          : out std_logic;
    nios_data     : out std_logic_vector(255 downto 0)

    );

end niosInterface;

architecture rtl of niosInterface is

signal control_store : std_logic := '0' ;


begin

  process (clk)
  begin
    if rising_edge(clk) then
      ctrl <= control_store;
      if reset_n = '0' then
```

```
        readdata <= (others => '0');
      control_store <= '0';
      else
        if chipselect = '1' then
                        if read = '1' then
                              readdata <= hardware_data;
                        elsif write = '1' then
                                    case address is
                                          when "00000" =>
                                                control_store <=
writedata(0);
                                          when "00001" =>
                                                nios_data(31 downto 0)
<= (writedata (31 downto 0));
                                          when "00010" =>
                                                nios_data(63 downto 32)
<= (writedata (31 downto 0));
                                          when "00011" =>
                                                nios_data(95 downto 64)
<= (writedata (31 downto 0));
                                          when "00100" =>
                                                nios_data(127 downto
96) <= (writedata (31 downto 0));
                                          when "00101" =>
                                                nios_data(159 downto
128) <= (writedata (31 downto 0));
                                          when "00110" =>
                                                nios_data(191 downto
160) <= (writedata (31 downto 0));
                                          when "00111" =>
                                                nios_data(223 downto
192) <= (writedata (31 downto 0));
                                          when "01000" =>
                                                nios_data(255 downto
224) <= (writedata (31 downto 0));
                                          when others =>
                                                control_store <= '0';
                                    end case;
                        end if;
                  end if;
            end if;
      end if;
  end process;

end rtl;
=================================================================
```

ray_FSM.vhd

```
=================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ray_FSM is
```

```
        port ( clk              : in  std_logic;
                              control         : in std_logic;
                              VGA_BLANK       : in std_logic;
                              wrfull          : in std_logic;
                              posX            : in unsigned(31 downto
    0);
                              posY            : in  unsigned (31 downto
    0);
                              countstep       : in unsigned (31 downto
    0);
                              colAddrIn       : in unsigned (9 downto
    0);
                              rayDirX         : in signed (31 downto
    0);
                              rayDirY         : in signed (31 downto
    0);
                              tmpPosXout      : out unsigned (31 downto
    0);
                              tmpPosYout      : out unsigned (31 downto
    0);
                              isSide          : out std_logic;
                              isSide2         : out std_logic;
                              bool            : out std_logic;
                              texNum          : out unsigned (3 downto
    0);
                              texNum2         : out unsigned (3 downto
    0);
                              texX            : out unsigned (31 downto
    0);
                              texX2           : out unsigned (31 downto
    0);
                              floorX          : out unsigned (31 downto
    0);
                              floorY          : out unsigned (31 downto
    0);
                              countout        : out unsigned (31 downto
    0);
                              countout2       : out unsigned (31 downto
    0);
                              line_minus_h    : out unsigned (31 downto
    0);
                              invline         : out unsigned (31 downto
    0);
                              line_minus_h2   : out unsigned (31 downto
    0);
                              invline2        : out unsigned (31 downto
    0);
                              invdist_out     : out unsigned (31 downto
    0);
                              drawStart       : out unsigned (31 downto
    0);
                              drawMid         : out unsigned (31 downto
    0);
                              drawEnd         : out unsigned (31 downto
    0);
                              colAddrOut      : out unsigned (9 downto
    0);
```

```
                              state_out       : out std_logic_vector
(11 downto 0);
                              WE              : out std_logic;
                              VGA_BLANK_OUT   : out std_logic;
                              ready           : out std_logic
);
end ray_FSM;

architecture imp of ray_FSM is

type rom_type is array(0 to 1023) of unsigned (3 downto 0);
constant MAP_ROM: rom_type := (
x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x
"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"
9",x"9",x"9",x"9",x"9",x"0",x"0",x"7",x"0",x"8",x"0",x"8",x"0",x"0",x"8
",x"7",x"0",x"2",x"5",x"2",x"6",x"2",x"5",x"2",x"0",x"6",x"0",x"0",x"0"
,x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"3",x"3",x"0",x"0",x"0",
x"0",x"0",x"8",x"8",x"4",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x
"0",x"0",x"0",x"0",x"0",x"5",x"0",x"0",x"6",x"0",x"9",x"9",x"0",x"0",x"
3",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0
",x"0",x"0",x"0",x"0",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9"
,x"9",x"2",x"3",x"3",x"0",x"0",x"0",x"0",x"0",x"8",x"8",x"4",x"0",x"0",
x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"6",x"0",x"0",x"0",x
"6",x"0",x"0",x"9",x"9",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"
8",x"4",x"0",x"0",x"0",x"0",x"0",x"6",x"6",x"6",x"0",x"6",x"7",x"0",x"0
",x"0",x"5",x"0",x"0",x"0",x"0",x"9",x"9",x"8",x"8",x"8",x"0",x"8",x"8"
,x"8",x"8",x"8",x"8",x"4",x"4",x"4",x"4",x"4",x"4",x"6",x"0",x"0",x"0",
x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"7",x"0",x"9",x"9",x"6",x"7",x
"7",x"0",x"7",x"7",x"7",x"7",x"0",x"8",x"0",x"8",x"0",x"8",x"0",x"8",x"
4",x"0",x"4",x"0",x"6",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9
",x"9",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"7",x"8",x"0",x"8",x"0",x"8"
,x"0",x"8",x"8",x"6",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"8",x"0",
x"8",x"0",x"0",x"9",x"9",x"2",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x
"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"7",x"0",x"
0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"0",x"0",x"0",x"0
",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"8",x"6",x"0",x"6",x"0"
,x"6",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"7",x"0",
x"0",x"0",x"0",x"0",x"0",x"7",x"8",x"0",x"8",x"0",x"8",x"0",x"8",x"8",x
"6",x"4",x"6",x"0",x"6",x"6",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"
9",x"9",x"2",x"7",x"7",x"0",x"7",x"7",x"7",x"7",x"8",x"8",x"4",x"0",x"6
",x"8",x"4",x"8",x"3",x"3",x"3",x"0",x"3",x"6",x"0",x"0",x"0",x"0",x"0"
,x"0",x"0",x"0",x"9",x"9",x"6",x"2",x"2",x"0",x"2",x"2",x"2",x"2",x"4",
x"6",x"4",x"0",x"0",x"6",x"0",x"6",x"3",x"0",x"0",x"0",x"0",x"0",x"0",x
"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"6",x"0",x"0",x"0",x"0",x"
0",x"2",x"2",x"4",x"0",x"0",x"0",x"0",x"0",x"0",x"4",x"3",x"0",x"0",x"0
",x"0",x"6",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"0",x"0"
,x"0",x"0",x"0",x"0",x"0",x"2",x"4",x"0",x"0",x"0",x"0",x"0",x"0",x"4",
x"3",x"0",x"0",x"0",x"0",x"0",x"0",x"3",x"2",x"0",x"4",x"3",x"0",x"0",x
"9",x"9",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"1",x"4",x"4",x"0",x"4",x"
4",x"6",x"0",x"6",x"3",x"3",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0
",x"0",x"0",x"0",x"9",x"9",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"2",x"2"
,x"2",x"0",x"2",x"2",x"2",x"6",x"6",x"0",x"0",x"5",x"0",x"5",x"0",x"0",
x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"7",x"0",x"0",x"0",x"0",x
"0",x"2",x"2",x"2",x"0",x"0",x"0",x"2",x"2",x"3",x"5",x"0",x"5",x"0",x"
0",x"0",x"5",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"0",x"0
",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0"
,x"0",x"0",x"0",x"0",x"5",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",
```

```
x"9",x"9",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"3",x"4",x"2",x"1",x"1",x
"2",x"0",x"4",x"4",x"3",x"1",x"2",x"2",x"0",x"0",x"0",x"0",x"0",x"0",x"
0",x"0",x"0",x"0",x"9",x"9",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"2",x"0
",x"0",x"0",x"0",x"0",x"2",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0"
,x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"2",x"7",x"2",
x"7",x"2",x"2",x"2",x"0",x"7",x"0",x"8",x"8",x"0",x"5",x"0",x"4",x"0",x
"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"
0",x"2",x"7",x"2",x"7",x"2",x"2",x"2",x"0",x"7",x"0",x"8",x"8",x"0",x"5
",x"0",x"4",x"0",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0"
,x"9",x"9",x"2",x"0",x"2",x"0",x"0",x"0",x"0",x"2",x"2",x"0",x"7",x"0",
x"8",x"8",x"0",x"5",x"0",x"4",x"0",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x
"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"2",x"0",x"2",x"0",x"0",x"0",x"
0",x"0",x"7",x"0",x"8",x"8",x"0",x"3",x"0",x"4",x"0",x"7",x"0",x"0",x"0
",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"2",x"0",x"2"
,x"0",x"2",x"2",x"2",x"0",x"2",x"0",x"8",x"8",x"0",x"5",x"0",x"4",x"0",
x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x
"0",x"2",x"0",x"2",x"0",x"0",x"0",x"1",x"0",x"2",x"0",x"3",x"3",x"0",x"
5",x"0",x"4",x"0",x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0
",x"9",x"9",x"2",x"0",x"2",x"0",x"2",x"7",x"2",x"0",x"2",x"0",x"1",x"0"
,x"8",x"8",x"0",x"3",x"0",x"4",x"0",x"7",x"0",x"0",x"0",x"0",x"0",x"0",
x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"2",x"0",x"2",x"7",x"1",x"0",x
"1",x"0",x"1",x"0",x"8",x"8",x"0",x"5",x"0",x"4",x"0",x"7",x"0",x"0",x"
0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"2",x"0",x"0",x"0",x"2
",x"7",x"2",x"1",x"2",x"0",x"7",x"0",x"0",x"0",x"0",x"5",x"0",x"0",x"0"
,x"7",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"9",x"9",x"9",
x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x
"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"9",x"
9",x"9"
);



type states is (A, B, C, D, E ,F, G, H, I,J,K,L);
signal state : states := A;

-- loop mod signals
signal controlprev : std_logic := '0';
signal count : unsigned (31 downto 0) := (others => '0');
signal count2 : unsigned (31 downto 0) := (others => '0');

signal countstep_sig : unsigned (31 downto 0):= x"00035a4b";
signal rayDirX_sig : signed (31 downto 0):= x"00035a4b";
signal rayDirY_sig : signed (31 downto 0 ):= x"fffdd183";
signal rayDirX_calc : signed (31 downto 0):= x"00035a4b";
signal rayDirY_calc : signed (31 downto 0 ):= x"fffdd183";

signal rayPosX : unsigned (31 downto 0):= x"05c00000";
signal rayPosY : unsigned (31 downto 0):= x"02e00000";

signal rayPosX2 : unsigned (31 downto 0):= x"05c00000";
signal rayPosY2 : unsigned (31 downto 0):= x"02e00000";

signal colAddr : unsigned (9 downto 0):= "0000000001";

signal tmpPosX : unsigned (31 downto 0) := x"05c00000";
signal tmpPosY : unsigned (31 downto 0) := x"02e00000";
```

```vhdl
signal mapSpot : unsigned (3 downto 0) :=(others => '0');
signal mapSpot2 : unsigned (3 downto 0) :=(others => '0');

signal countshift : unsigned (31 downto 0):=(others => '0');
signal lineheight : unsigned(31 downto 0):=(others => '0');
signal invlineheight : unsigned(31 downto 0):=(others => '0');

signal lineheight2 : unsigned(31 downto 0):=(others => '0');
signal invlineheight2 : unsigned(31 downto 0):=(others => '0');

signal invdist : unsigned(31 downto 0):=(others => '0');
signal remainder_line : unsigned(31 downto 0):=(others => '0');
signal remainder_invline : unsigned(31 downto 0):=(others => '0');

signal remainder_line2 : unsigned(31 downto 0):=(others => '0');
signal remainder_invline2 : unsigned(31 downto 0):=(others => '0');

signal remainder_invdist : unsigned(31 downto 0):=(others => '0');
signal inc : unsigned(4 downto 0):=(others => '0');

signal inc_limit1 : unsigned(11 downto 0) := "111111111111";
signal inc_limit2 : unsigned(5 downto 0) := "111111";

signal bitselect : unsigned(31 downto 0):=(others => '0');
signal tmpCount : unsigned (31 downto 0):=(others => '0');
signal tmplineNum : unsigned (31 downto 0):=(others => '0');
signal tmpinvDistNum : unsigned (31 downto 0):=(others => '0');

signal tmpCount2 : unsigned (31 downto 0):=(others => '0');

constant line_numerator : unsigned(31 downto 0) := x"78000000";
constant screenHeight : unsigned(31 downto 0) := x"000001E0";
constant halfscreenHeight : unsigned(31 downto 0) := x"000000F0";
constant invdist_numerator : unsigned(31 downto 0) := x"01000000";


begin

process (clk) --Sequential process

  variable drawStartTmp           : unsigned (31 downto 0);
  variable drawMidTmp             : unsigned (31 downto 0);
  variable drawEndTmp             : unsigned (31 downto 0);
  variable tmp_rayPosX            : unsigned (31 downto 0);
  variable tmp_rayPosY            : unsigned (31 downto 0);

  variable tmp_rayPosX2           : unsigned (31 downto 0);
  variable tmp_rayPosY2           : unsigned (31 downto 0);

  variable addr                   : unsigned (9 downto 0);
  variable addr2                  : unsigned (9 downto 0);

  variable remainder_line_var     : unsigned (31 downto 0);
  variable remainder_invline_var  : unsigned (31 downto 0);

  variable remainder_line_var2    : unsigned (31 downto 0);
  variable remainder_invline_var2 : unsigned (31 downto 0);
```

```
  variable remainder_invdist_var  : unsigned (31 downto 0);

begin

if rising_edge(clk) then
      colAddrOut <= colAddr;
      controlprev <= control;

 case state is
            when A =>
                          inc_limit1 <= "111111111111";
                          inc_limit2 <= "111111";
                          state_out <= "100000000000";
                          ready <= '1';
                          WE<='0';
                          VGA_BLANK_OUT <= '0';

                          if (controlprev = '0' and control = '1') then
                                     count <= x"00000000";
                                     count2 <= x"00000000";

                                     countstep_sig <= countstep;
                                     rayDirX_sig <= rayDirX;
                                     rayDirY_sig <= rayDirY;
                                     rayDirX_calc <= rayDirX;
                                     rayDirY_calc <= rayDirY;

                                     rayPosX <= posX;
                                     rayPosY <= posY;

                                     rayPosX2 <= posX;
                                     rayPosY2 <= posY;

                                     tmpPosX <= posX;
                                     tmpPosY <= posY;

                                     tmp_rayPosX  := posX;
                                     tmp_rayPosY  := posY;
                                     addr := ((tmp_rayPosX (26 downto
22)) & "00000" ) + (tmp_rayPosY (31 downto 22));

                                     tmp_rayPosX2  := posX;
                                     tmp_rayPosY2  := posY;
                                     addr2 := ((tmp_rayPosX2 (26 downto
22)) & "00000" )  + (tmp_rayPosY2 (31 downto 22));


                                     mapSpot <=
MAP_ROM(to_integer(addr));
                                     mapSpot2 <=
MAP_ROM(to_integer(addr2));
                                     state <= B;

                                     colAddr <= colAddrIn;

                          else
                                           state <= A;
```

```
                        end if;

            when B =>
                        state_out <= "010000000000";
                        ready <= '0';
                        WE<='0';
                        VGA_BLANK_OUT <= '0';

                        inc_limit1 <= inc_limit1 - 1;
                        if (mapSpot2 < x"5" and inc_limit1 >
"000000000000" ) then

                                    if (mapSpot = x"0") then
                                                count <= count +
countstep_sig;
                                                rayPosX <=
unsigned(signed(rayPosX) +   rayDirX_sig);
                                                rayPosY <=
unsigned(signed(rayPosY) + rayDirY_sig);
                                    end if;

                                    count2 <= count2 + countstep_sig;
                                    rayPosX2 <=
unsigned(signed(rayPosX2) + rayDirX_sig);
                                    rayPosY2 <=
unsigned(signed(rayPosY2) +   rayDirY_sig);


                                    tmp_rayPosX  :=
unsigned(signed(rayPosX) +      rayDirX_sig);
                                    tmp_rayPosY  :=
unsigned(signed(rayPosY) + rayDirY_sig);
                                    addr := ((tmp_rayPosX (26 downto
22)) & "00000" ) + (tmp_rayPosY (31 downto 22));

                                    tmp_rayPosX2  :=
unsigned(signed(rayPosX2) +    rayDirX_sig);
                                    tmp_rayPosY2  :=
unsigned(signed(rayPosY2) + rayDirY_sig);
                                    addr2 := ((tmp_rayPosX2 (26 downto
22)) & "00000" )  + (tmp_rayPosY2 (31 downto 22));


                                    mapSpot <=
MAP_ROM(to_integer(addr));
                                    mapSpot2 <=
MAP_ROM(to_integer(addr2));
                                    state <= B;
                        else
                                          state <= C;
                        end if;

            when C =>
                        state_out <= "001000000000";
                        ready <= '0';
                        WE<='0';
                        VGA_BLANK_OUT <= '0';
```

```
                          state<= D;

                          --decrement variables to increase precision
                          countstep_sig <= "00000" & countstep_sig(31
downto 5);

                          -- if negative shift in 1's
                          if ( rayDirX_sig(31 downto 31) = "1") then
                                     rayDirX_sig <= "11111" &
rayDirX_sig (31 downto 5);
                          else
                                     rayDirX_sig <= "00000"  &
rayDirX_sig(31 downto 5);
                          end if;

                          if ( rayDirY_sig(31 downto 31) = "1") then
                                     rayDirY_sig <= "11111" &
rayDirY_sig (31 downto 5);
                          else
                                     rayDirY_sig <= "00000"  &
rayDirY_sig(31 downto 5);
                          end if;

            when D =>
                          state_out <= "000100000000";
                          ready <= '0';
                          WE<='0';
                          VGA_BLANK_OUT <= '0';

                          inc_limit2 <= inc_limit2 - 1;

                          if( (mapSpot = x"0" and mapSpot2 < x"5") or
inc_limit2 = "000000") then
                                           state <= E;
                          else
                                     if (mapSpot > x"0") then
                                                     count <= count -
countstep_sig;
                                                     rayPosX <=
unsigned(signed(rayPosX) -    rayDirX_sig);
                                                     rayPosY <=
unsigned(signed(rayPosY) -   rayDirY_sig);

                                     end if;

                                     if (mapSpot2 > x"4") then

                                          count2 <= count2 -
countstep_sig;
                                    rayPosX2 <=
unsigned(signed(rayPosX2) - rayDirX_sig);
                                    rayPosY2 <=
unsigned(signed(rayPosY2) - rayDirY_sig);

                                     end if;
```

```
                              tmp_rayPosX  := unsigned(signed(rayPosX)
- rayDirX_sig);
                              tmp_rayPosY  := unsigned(signed(rayPosY)
- rayDirY_sig);
                            addr := ((tmp_rayPosX (26 downto 22)) &
"00000" ) + (tmp_rayPosY (31 downto 22));

                                    tmp_rayPosX2  :=
unsigned(signed(rayPosX2) -    rayDirX_sig);
                                     tmp_rayPosY2  :=
unsigned(signed(rayPosY2) - rayDirY_sig);
                                     addr2 := ((tmp_rayPosX2 (26 downto
22)) & "00000" )  + (tmp_rayPosY2 (31 downto 22));
                                     mapSpot <=
MAP_ROM(to_integer(addr));
                                     mapSpot2 <=
MAP_ROM(to_integer(addr2));
                                     state <= D;

                          end if;


            when E =>
                          state_out <= "000010000000";
                          ready <= '0';
                          WE<='0';
                          VGA_BLANK_OUT <= '0';

                          count <= count + countstep_sig;
                          rayPosX <= unsigned(signed(rayPosX) +
rayDirX_sig);
                          rayPosY <= unsigned(signed(rayPosY) +
rayDirY_sig);

                          count2 <= count2 + countstep_sig;
                          rayPosX2 <= unsigned(signed(rayPosX2) +
rayDirX_sig);
                          rayPosY2 <= unsigned(signed(rayPosY2) +
rayDirY_sig);

                  tmp_rayPosX := unsigned(signed(rayPosX) +
rayDirX_sig);
                  tmp_rayPosY := unsigned(signed(rayPosY) +
rayDirY_sig);
                          addr := ((tmp_rayPosX (26 downto 22)) &
"00000" ) + (tmp_rayPosY (31 downto 22));

                          tmp_rayPosX2  := unsigned(signed(rayPosX2) +
rayDirX_sig);
                          tmp_rayPosY2  := unsigned(signed(rayPosY2) +
rayDirY_sig);
                          addr2 := ((tmp_rayPosX2 (26 downto 22)) &
"00000" )  + (tmp_rayPosY2 (31 downto 22));

                          mapSpot <= MAP_ROM(to_integer(addr));
                          mapSpot2 <= MAP_ROM(to_integer(addr2));
```

```
                          state <= F;

            when F =>
                        state_out <= "000001000000";
                          ready <= '0';
                          WE<='0';
                          VGA_BLANK_OUT <= '0';

                          inc <= "11111";

                          countshift <= "0000000000"      & count(31
downto 10);
                          lineheight <= x"00000000";
                          invlineheight <= x"00000000";
                          invdist <= x"00000000";

                          lineheight2 <= x"00000000";
                          invlineheight2 <= x"00000000";

                          remainder_line <= x"00000000";
                          remainder_invline <= x"00000000";
                          remainder_invdist <= x"00000000";

                          remainder_line2 <= x"00000000";
                          remainder_invline2 <= x"00000000";

                          bitselect <= x"80000000";
                          tmplineNum <= line_numerator;
                          tmpCount <=  "0"&count(31 downto 1);
                          tmpinvDistNum <= invdist_numerator;

                          tmpCount2 <=  "0"&count2(31 downto 1);

                          state <= G;

            when G =>
                          state_out <= "000000100000";
                          ready <= '0';
                          WE<='0';
                          VGA_BLANK_OUT <= '0';

                          if (inc =  "00000") then
                                            state <= H;
                          else
                                            state <= G;
                          end if;

                          remainder_line_var := (remainder_line (30
downto 0))& tmplineNum(31 downto 31);
                          remainder_invline_var :=
(remainder_invline(30 downto 0) )& tmpCount(31 downto 31);
                          remainder_invdist_var :=
(remainder_invdist(30 downto 0))& tmpinvDistNum(31 downto 31);

                          remainder_line_var2 := (remainder_line2 (30
downto 0))& tmplineNum(31 downto 31);
```

```
                          remainder_invline_var2 :=
(remainder_invline2(30 downto 0) )& tmpCount2(31 downto 31);

                          tmplineNum <= tmplineNum(30 downto 0) & "0";
                          tmpCount <= tmpCount(30 downto 0) & "0";
                          tmpCount2 <= tmpCount2(30 downto 0) & "0";
                          tmpinvDistNum <= tmpinvDistNum(30 downto 0) &
"0";

                          bitselect <= "0" & bitselect(31 downto 1);

                          if (remainder_line_var >= count) then
                                          remainder_line <=
remainder_line_var - count;
                                          lineheight <= lineheight +
bitselect;
                          else
                                          remainder_line <=
remainder_line_var;
                          end if;

                          if (remainder_invline_var >= screenHeight)
then
                                          remainder_invline <=
remainder_invline_var - screenHeight;
                                          invlineheight <=
invlineheight + bitselect;
                          else
                                          remainder_invline <=
remainder_invline_var;
                          end if;

                          if (remainder_invdist_var >= countshift) then
                                          remainder_invdist <=
remainder_invdist_var - countshift;
                                          invdist <=invdist +
bitselect;
                          else
                                          remainder_invdist <=
remainder_invdist_var;
                          end if;
                          if (remainder_line_var2 >= count2) then
                                          remainder_line2 <=
remainder_line_var2 - count2;
                                          lineheight2 <= lineheight2 +
bitselect;
                          else
                                          remainder_line2 <=
remainder_line_var2;
                          end if;

                   if (remainder_invline_var2 >= screenHeight) then
                        remainder_invline2 <= remainder_invline_var2 -
screenHeight;
                        invlineheight2 <= invlineheight2 + bitselect;
                          else
```

```
                                    remainder_invline2 <=
remainder_invline_var2;
                        end if;

                        inc <= inc - 1;

            when H =>
                        state_out <= "000000010000";
                        ready <= '0';
                        WE<='0';
                        VGA_BLANK_OUT <= '0';

                        if (mapSpot > x"4") then
                                    bool <= '1';
                                    drawStartTmp :=
halfscreenHeight - (lineheight(30 downto 0)& "0")  - ("0" &
lineheight(31 downto 1));
                                    texNum <= mapSpot - 5;
                        else
                                    bool <= '0';
                                    drawStartTmp :=
halfscreenHeight - (lineheight2(30 downto 0) & "0") -("0" &
lineheight2(31 downto 1));
                                    texNum <= mapSpot - 1;
                        end if;

                        texNum2 <= mapSpot2 - 5;

            drawMidTmp  := halfscreenHeight - ("0" & lineheight(31
downto 1));
            drawEndTmp  := halfscreenHeight + ( "0" & lineheight(31
downto 1));

                        if (drawStartTmp >= screenHeight) then
                                    drawStart <= x"00000000";
                        else
                                    drawStart <= drawStartTmp;
                        end if;

                        if (drawMidTmp >= screenHeight) then
                                    drawMid <= x"00000000";
                        else
                                    drawMid <= drawMidTmp;
                        end if;

                        if (drawEndTmp >= screenHeight) then
                                    drawEnd <= screenHeight -1;
                        else
                                    drawEnd <= drawEndTmp;
                        end if;

                        line_minus_h <= lineheight - screenHeight;
                        invline <= invlineheight;
                        invdist_out <= invdist;
                        line_minus_h2 <= lineheight2 - screenHeight;
                        invline2 <= invlineheight2;
                        state <= I;
```

```
            when I =>
                        state_out <= "000000001000";
                        ready <= '0';
                        WE<='0';
                        VGA_BLANK_OUT <= '0';

                        if (wrfull = '1') then
                                state <= I;
                        else
                                state <= J;
                        end if;
            when J =>
                        state_out <= "000000000100";
                        ready <= '0';
                        VGA_BLANK_OUT <= '0';
                        if (colAddr >= "1001111111") then
                                state <= K;
                                WE<='0';
                        else
                                WE<='1';
                                state <= A;
                        end if;

            when K =>
                        state_out <= "000000000010";
                        ready <= '0';
                        if (VGA_BLANK = '1') then
                                state <= A;
                                WE<='1';
                                VGA_BLANK_OUT <= '1';
                        else
                                VGA_BLANK_OUT <= '0';
                            state <= K;
                                WE<='0';
                        end if;

            when others =>
                        state_out <= "111111111111";
                        state <= A;

            end case;

  end if;
end process;
------------------------------------------------------------


process (count,count2,mapSpot,mapSpot2, rayPosX,
rayPosY,rayPosX2,rayPosY2,tmpPosX ,tmpPosY ,rayDirX_calc, rayDirY_calc)

  variable mapX : unsigned (31 downto 0);
  variable mapX2 : unsigned (31 downto 0);

  variable mapY : unsigned (31 downto 0);
  variable mapY2 : unsigned (31 downto 0);
```

```
  variable checkSide : std_logic;
  variable checkSide2 : std_logic;

  variable wallX : unsigned (31 downto 0);
  variable wallX2 : unsigned (31 downto 0);

  variable tmpTexX : unsigned (31 downto 0);
  variable tmpTexX2 : unsigned (31 downto 0);

  variable floorXvar : unsigned (31 downto 0);
  variable floorYvar : unsigned (31 downto 0);

  begin

                   if rayDirX_calc < 0 then
                                   mapX := ((rayPosX srl 22) + 1) sll
22;
                                   mapX2 := ((rayPosX2 srl 22) + 1)
sll 22;
                   else
                                   mapX := (rayPosX srl 22) sll 22;
                                   mapX2 := (rayPosX2 srl 22) sll 22;
                   end if;

                   if rayDirY_calc < 0 then
                                   mapY := ((rayPosY srl 22) + 1) sll
22;
                                   mapY2 := ((rayPosY2 srl 22) + 1)
sll 22;
                   else
                                   mapY := (rayPosY srl 22) sll 22;
                                   mapY2 := (rayPosY2 srl 22) sll 22;
                   end if;

                   --Calculate distance of perpendicular ray (oblique
distance will give fisheye effect!)
                   checkSide := '0';
                   checkSide2 := '0';

                   if rayDirX_calc > 0 and rayDirY_calc > 0 then
                                   if (rayPosX - mapX) < (rayPosY -
mapY) then
                                                  checkSide := '1';
                                   end if;
                                   if (rayPosX2 - mapX2) < (rayPosY2 -
mapY2) then
                                                  checkSide2 := '1';
                                   end if;

                   elsif rayDirX_calc > 0 and rayDirY_calc < 0 then

                                   if (rayPosX - mapX) < (mapY -
rayPosY) then
                                                  checkSide := '1';
                                   end if;
```

```
                                    if (rayPosX2 - mapX2) < (mapY2 -
rayPosY2) then
                                                  checkSide2 := '1';
                                    end if;

                  elsif rayDirX_calc < 0 and rayDirY_calc > 0 then

                                    if (mapX - rayPosX) < (rayPosY -
mapY) then
                                                  checkSide := '1';
                                    end if;

                                    if (mapX2 - rayPosX2) < (rayPosY2 -
mapY2) then
                                                  checkSide2 := '1';
                                    end if;

                  elsif rayDirX_calc < 0 and rayDirY_calc < 0 then

                                    if (mapX - rayPosX) < (mapY -
rayPosY) then
                                                  checkSide := '1';
                                    end if;

                                    if (mapX2 - rayPosX2) < (mapY2 -
rayPosY2) then
                                                  checkSide2 := '1';
                                    end if;
                  end if;


                  if checkSide = '0' then wallX := rayPosX;
                  else wallX := rayPosY;
                  end if;

                  if checkSide2 = '0' then wallX2 := rayPosX2;
                  else wallX2 := rayPosY2;
                  end if;


                  wallX := wallX - ((wallX srl 22) sll 22);
                  wallX2 := wallX2 - ((wallX2 srl 22) sll 22);

                  --x coordinate on the texture
                  tmpTexX := wallX srl 16;
                  tmpTexX2 := wallX2 srl 16;

                  if ((checkSide = '1') and (rayDirX_calc > 0)) or
((checkSide = '0') and (rayDirY_calc < 0)) then
                                    tmpTexX := 64 - tmpTexX - 1;
                  end if;

                  if ((checkSide2 = '1') and (rayDirX_calc > 0)) or
((checkSide2 = '0') and (rayDirY_calc < 0)) then
                                    tmpTexX2 := 64 - tmpTexX2 - 1;
                  end if;
```

```
                    texX <= tmpTexX;
                    texX2 <= tmpTexX2;

                    --x, y position of the floor texel at the bottom of
the wall
                    if (checkSide = '1') and (rayDirX_calc > 0) then
                                  floorXvar := (rayPosX srl 22) sll
22;
                                  floorYvar := ((rayPosY srl 22) sll
22) + wallX;
                    elsif (checkSide = '1') and (rayDirX_calc < 0) then
                                  floorXvar := ((rayPosX srl 22) sll
22) + "10000000000000000000000";
                                  floorYvar := ((rayPosY srl 22) sll
22) + wallX;
                    elsif (checkSide = '0') and (rayDirY_calc > 0) then
                                  floorXvar := ((rayPosX srl 22) sll
22) + wallX;
                                  floorYvar := (rayPosY srl 22) sll
22;
                    else
                                  floorXvar := ((rayPosX srl 22) sll
22) + wallX;
                                  floorYvar := ((rayPosY srl 22) sll
22) + "10000000000000000000000";
                    end if;


                    isSide <= checkSide;
                    isSide2 <= checkSide2;
                    countout <= count;
                    countout2 <= count2;

                    floorX <= floorXvar srl 10;
                    floorY <= floorYvar srl 10;

                    tmpPosXout <= tmpPosX srl 10;
                    tmpPosYout <= tmpPosY srl 10;

  end process;

end imp;
========================================================================
```

skygen.vhd

```
========================================================================
-----------------------------------------------------------------------
--
-- Sky Generation
--
-- Wei-Hao,
-----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity skygen is

  port (
    signal reset_n : in std_logic;
    signal clk   : in std_logic;                    -- Should be 50 MHz

   signal read       : in  std_logic;
   signal write      : in  std_logic;
   signal chipselect : in  std_logic;
   signal address    : in  std_logic_vector(17 downto 0);
   signal readdata   : out std_logic_vector(15 downto 0);
   signal writedata  : in  std_logic_vector(15 downto 0);
   signal byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N            : out std_logic;

        signal Cur_Row_in       : in std_logic_vector (9 downto 0);
        signal FB_angle_in        : in std_logic_vector (9 downto 0);
        signal Sky_pixel  : out std_logic_vector (7 downto 0);
          signal Sram_mux_out         : out std_logic

    );
end skygen;

ARCHITECTURE SYN OF skygen IS

        SIGNAL    sram_mux                : std_logic;
--    SIGNAL      cpu_data          : STD_LOGIC_VECTOR (15 DOWNTO 0);
--    SIGNAL      cpu_addr          : STD_LOGIC_VECTOR (16 DOWNTO 0) ;
--      SIGNAL  cpu_ub, cpu_lb;
--      SIGNAL  cpu_we, cpu_ce, cpu_oe;

      SIGNAL  Cur_Row                 : unsigned (9 downto 0);
        SIGNAL  FB_angle                : unsigned (9 downto 0);

      SIGNAL  vga_addr_lsb            : std_logic;
      --SIGNAL  vga_addr_tmp        : unsigned (18 downto 0) ;
      SIGNAL  vga_addr        : STD_LOGIC_VECTOR (17 downto 0) ;

BEGIN
  -- Communicate with MEM
  MEM : process (clk)
  begin
  if rising_edge (clk) then
      if reset_n = '0' then
                sram_mux <= '0';
      else
            if chipselect = '1' then
                  if write = '1' then
                        if address = "111111111111111111" then
                              sram_mux <= writedata (0);
```

```
                      end if;
                end if;
          end if;
      end if;
  end if;
  end process MEM;

        --Cur_Row <=  unsigned(Cur_Row_in(9 downto 0));
        --FB_angle <= unsigned(FB_angle_in(9 downto 0));

      addrGen : process (cur_row_in,FB_angle_in)
      variable vga_addr_tmp : unsigned(18 downto 0);

      begin
                vga_addr_tmp := unsigned(Cur_Row_in(9 downto
0)&"000000000") + unsigned(FB_angle_in(9 downto 1));
         vga_addr_lsb <= FB_angle_in(0);
                vga_addr <= std_logic_vector(vga_addr_tmp(17 downto
0));
      end process addrGen;

      skyPixelGen : process (SRAM_DQ, vga_addr_lsb)

      begin
            if (vga_addr_lsb = '0') then
                  Sky_pixel <= SRAM_DQ(7 downto 0);
            else
         Sky_pixel <= SRAM_DQ(15 downto 8);
            end if;

      end process skyPixelGen;



  SRAM_DQ <= writedata when write = '1' and sram_mux = '0' else
             (others => 'Z');

  SRAM_ADDR <= address           when sram_mux = '0' else
             vga_addr            when sram_mux = '1' else
             (others => '0');

  SRAM_UB_N <= not byteenable(1) when sram_mux = '0' else
            '0';

  SRAM_LB_N <= not byteenable(0) when sram_mux = '0' else
            '0';

  SRAM_WE_N <= not write         when sram_mux = '0' else
            '1';
  SRAM_CE_N <= not chipselect    when sram_mux = '0' else
            '0';
  SRAM_OE_N <= not read          when sram_mux = '0' else
            '0';

  readdata <=  SRAM_DQ;

  --Sky_pixel<=  (SRAM_DQ(7 downto 0)) when vga_addr_lsb = '0' else
```

```
    --          (SRAM_DQ(15 downto 8));

  Sram_mux_out <= sram_mux;

END SYN;
========================================================================
```

sound_controller.vhd

```
========================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sound_Controller is
port (
      -- Avalon:
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(7 downto 0);
    readdata   : out std_logic_vector(7 downto 0);
    writedata  : in  std_logic_vector(7 downto 0);
      -- connection to the component inside:
       irq :            OUT STD_LOGIC ;
      AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
      AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
      AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
      AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
      AUD_BCLK     : inout std_logic ; --    Audio CODEC Bit-Stream
Clock
      AUX_XCK     : out std_logic;  --    Audio CODEC Bit-Stream Clock
      led :              out std_LOGIC_vector(15 downto 0)
  );
end  sound_Controller;

architecture ar of sound_Controller is


 --FSM
 type states is (IDLE,WAITING);
 signal state : states:=IDLE;
 --Audio Interface
 signal audio_clock : unsigned(1 downto 0) := "00";
 signal indata : std_logic_vector (7 downto 0) ;
signal data_to_music : std_logic_vector (15 downto 0);
signal we,inter:std_logic;
 signal audio_request : std_logic;
 signal counter : integer:=0;

      component de2_wm8731_audio is port(
                clk : in std_logic;       --  Audio CODEC Chip Clock
AUD_XCK (18.43 MHz)
```

```
             reset_n : in std_logic;
                test_mode : in std_logic;       --    Audio CODEC
controller test mode
                audio_request : out std_logic;  --    Audio controller
request new data
                data : in std_logic_vector(15 downto 0);

                -- Audio interface signals
                AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC
LR Clock
                AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC
Data
                AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC
LR Clock
                AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC
Data
                AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-
Stream Clock


            );
      end component de2_wm8731_audio;

begin


  V1: de2_wm8731_audio port map (
    clk => audio_clock(1),
    reset_n => reset_n,
    test_mode => '0',                    -- Don t output a sine wave
    audio_request => audio_request,
    data => data_to_music,

    -- Audio interface signals
    AUD_ADCLRCK  => AUD_ADCLRCK,
    AUD_ADCDAT   => AUD_ADCDAT,
    AUD_DACLRCK  => AUD_DACLRCK,
    AUD_DACDAT   => AUD_DACDAT,
    AUD_BCLK     => AUD_BCLK
       --AUD_DACDAT   => led
  );

--Audio clock generation
process (clk)
begin
  if rising_edge(clk) then
--    we      <= '1' when chipselect = '1'and write ='1'  else '0' ;
      data_to_music <= indata & x"00"; --Since the audio is on 16 bits,
we pad the data with 0s
      --led <= data_to_music;
    audio_clock <= audio_clock + "1";
  end if;
end process;

AUX_XCK<=audio_clock(1);

--Combinational process
```

```
process (clk)
begin
      --By default reset the interuption signal

--    next_state <= state;
if (rising_edge(clk)) then

      if reset_n = '0' then
            state <= IDLE;
            irq<= '0';
            led <= "0000000000000000";
      end if;

      case state is

      --Waiting for a note request
      when IDLE =>
            led <= "1111100000000000";
            indata <=indata;
            if (audio_request = '1')      then
                  state <= WAITING;
                  irq <= '1';
            else
                  irq <='0';
                  state <= IDLE;
            end if;


      --A request has come. wait for writing
      when WAITING =>
      led <= "0000000000001111";
            --irq <='0';
            if (write ='1' and chipselect = '1') then       --
chipselect = '1' and write ='1'
                  indata <=writedata;
                  state <= IDLE;
                  irq <= '0';
            else
                  --if(counter=1000) then counter<=0;
                  --else counter<=counter +1;
                  --end if;
                  --if(counter=0) then
                        state <=WAITING;
                        irq<= '1';
                  --end if;
            end if;

      when others =>
            irq <= '0';
            state <= IDLE;

      end case;
end if;

end process;
end architecture;
```

==

tex_gen.vhd

==

```
----------------------------------------------------------------
--
-- Texture Generation
--
-- Edward Garcia
-- ewg2115@columbia.edu
--
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tex_gen is

  port (
    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 50 MHz
      -- clk25 : in std_logic;
       side1   : in std_logic;
       side2   : in std_logic;
       bool    : in std_logic;
       boolOut    : out std_logic;
       Cur_Row     : in unsigned (11 downto 0);
       Row_End     : in unsigned (8 downto 0);
       Row_Mid        : in unsigned (8 downto 0);
       texNum         : in unsigned (3 downto 0);
       texNum2        : in unsigned (3 downto 0);
       invLineHeight : in unsigned (17 downto 0);
       line_minus_h  : in signed (17 downto 0);
       invLineHeight2 : in unsigned (17 downto 0);
       line_minus_h2  : in signed (17 downto 0);
       texX                 : in unsigned (5 downto 0);
       texX2                : in unsigned (5 downto 0);
       floorX           : in unsigned (17 downto 0);
       floorY           : in unsigned (17 downto 0);
       tmpPosX          : in unsigned (17 downto 0);
    tmpPosY       : in unsigned (17 downto 0);
       invDistWall      : in unsigned (11 downto 0);
       y                    : in unsigned (8 downto 0);
       textureIndexOut: out unsigned (11 downto 0);
       SideOut         : out std_logic;
       texNumOut      : out unsigned (3 downto 0);
       texNum2Out        : out unsigned (3 downto 0);

--     Row_Start  : out unsigned (8 downto 0);
--    Row_End      : out unsigned (8 downto 0);
       tex_addr_out     : out unsigned (13 downto 0)
    );

end tex_gen;
```

```
architecture rtl of tex_gen is

type rom_type is array(0 to 239) of unsigned (15 downto 0);
constant DIVTABLE: rom_type := (
      x"1000",x"1011",x"1022",x"1033",x"1045",x"1057",x"1069",x"107b",x
"108d",x"109f",x"10b2",x"10c4",x"10d7",x"10ea",x"10fd",x"1111",x"1124",
x"1138",x"114c",x"1160",x"1174",x"1188",x"119d",x"11b2",x"11c7",x"11dc"
,x"11f1",x"1207",x"121c",x"1232",x"1249",x"125f",x"1276",x"128c",x"12a4
",x"12bb",x"12d2",x"12ea",x"1302",x"131a",x"1333",x"134b",x"1364",x"137
e",x"1397",x"13b1",x"13cb",x"13e5",x"1400",x"141a",x"1435",x"1451",x"14
6c",x"1488",x"14a5",x"14c1",x"14de",x"14fb",x"1519",x"1537",x"1555",x"1
573",x"1592",x"15b1",x"15d1",x"15f1",x"1611",x"1632",x"1653",x"1674",x"
1696",x"16b8",x"16db",x"16fe",x"1721",x"1745",x"176a",x"178e",x"17b4",x
"17d9",x"1800",x"1826",x"184d",x"1875",x"189d",x"18c6",x"18ef",x"1919",
x"1943",x"196e",x"1999",x"19c5",x"19f2",x"1a1f",x"1a4d",x"1a7b",x"1aaa"
,x"1ada",x"1b0a",x"1b3b",x"1b6d",x"1ba0",x"1bd3",x"1c07",x"1c3c",x"1c71
",x"1ca8",x"1cdf",x"1d17",x"1d50",x"1d89",x"1dc4",x"1e00",x"1e3c",x"1e7
9",x"1eb8",x"1ef7",x"1f38",x"1f79",x"1fbc",x"2000",x"2044",x"208a",x"20
d2",x"211a",x"2164",x"21af",x"21fb",x"2249",x"2298",x"22e8",x"233a",x"2
38e",x"23e3",x"2439",x"2492",x"24ec",x"2548",x"25a5",x"2605",x"2666",x"
26c9",x"272f",x"2796",x"2800",x"286b",x"28d9",x"294a",x"29bd",x"2a32",x
"2aaa",x"2b25",x"2ba2",x"2c23",x"2ca6",x"2d2d",x"2db6",x"2e43",x"2ed4",
x"2f68",x"3000",x"309b",x"313b",x"31de",x"3286",x"3333",x"33e4",x"349a"
,x"3555",x"3615",x"36db",x"37a6",x"3878",x"3950",x"3a2e",x"3b13",x"3c00
",x"3cf3",x"3def",x"3ef3",x"4000",x"4115",x"4234",x"435e",x"4492",x"45d
1",x"471c",x"4873",x"49d8",x"4b4b",x"4ccc",x"4e5e",x"5000",x"51b3",x"53
7a",x"5555",x"5745",x"594d",x"5b6d",x"5da8",x"6000",x"6276",x"650d",x"6
7c8",x"6aaa",x"6db6",x"70f0",x"745d",x"7800",x"7bde",x"8000",x"8469",x"
8924",x"8e38",x"93b1",x"9999",x"a000",x"a6f4",x"ae8b",x"b6db",x"c000",x
"ca1a",x"d555",x"e1e1",x"f000",x"0000",x"1249",x"2762",x"4000",x"5d17",
x"8000",x"aaaa",x"e000",x"2492",x"8000",x"0000",x"c000",x"0000",x"8000"
,x"0000"
);
--  constant SCREENHEIGHT        : unsigned(8 downto 0) := "111100000";
--480
--  constant SCREENHEIGHT_HALF   : unsigned(8 downto 0) := "011110000";
--240
--  constant SCREENHEIGHTMINUS1  : unsigned(8 downto 0) := "111111110";
--470

      signal y_local : unsigned (8 downto 0);
      signal cur_row_local : unsigned (11 downto 0);
      signal tex_addr : unsigned (13 downto 0);
      signal clk_enable : std_logic := '0';
      signal inA_0 : std_logic_vector(17 downto 0) := (others => '0');
      signal inB_0 : std_logic_vector(17 downto 0) := (others => '0');
      signal inA_1 : std_logic_vector(17 downto 0) := (others => '0');
      signal inB_1 : std_logic_vector(17 downto 0) := (others => '0');
      signal result_0 : std_logic_vector(35 downto 0);
      signal result_1 : std_logic_vector(35 downto 0);
      signal sideSig : std_logic;
      signal boolSig : std_logic := '0';
      signal texNumSig : unsigned (3 downto 0);
      signal texNum2Sig : unsigned (3 downto 0);
--    signal tmp : unsigned (12 downto 0);
--    signal floorXprev : unsigned (17 downto 0);
```

```
--    signal floorYprev : unsigned (17 downto 0 );
--   signal tmpPosXprev : unsigned (17 downto 0);
--    signal tmpPosYprev : unsigned (17 downto 0);
--    signal weight : unsigned (27 downto 0);
--    signal texY_0 : unsigned (35 downto 0);
--    signal texY_1 : unsigned (35 downto 0);
--
--    signal texXprev : unsigned (5 downto 0);
--    signal texX2prev : unsigned (5 downto 0);
--
--   signal texNumprev : unsigned (3 downto 0);
--    signal texNum2prev : unsigned (3 downto 0);
--
--    signal texNumprev_2 : unsigned (3 downto 0);
--    signal texNum2prev_2 : unsigned (3 downto 0);
--
--    signal side1prev : std_logic;
--    signal side2prev :std_logic;
--    signal boolprev  : std_logic;
--
--    signal row_Mid_prev : unsigned (8 downto 0);
--    signal row_End_prev : unsigned ( 8 downto 0);

begin
   S0: entity work.mult port map (
            dataa       => inA_0,
            datab       => inB_0,
            result      => result_0
  );

   S1: entity work.mult port map (
            dataa       => inA_1,
            datab       => inB_1,
            result      => result_1
  );

      process (floorX, floorY, tmpPosX, tmpPosY, invDistWall, y
,row_end , cur_row, texX,texNum,
line_minus_h,invLineHeight,texX2,texNum2, line_minus_h2,invLineHeight2,
bool, side1, side2, result_0,result_1 ,row_Mid)

      --process (clk)

      variable texY_0 : unsigned (35 downto 0);
      variable texY_1 : unsigned (35 downto 0);
      variable currentDist : unsigned (15 downto 0);
      variable weight:  unsigned (27 downto 0);
      variable tmp : unsigned (12 downto 0);
      variable currentFloorX : unsigned (30 downto 0);
      variable currentFloorY : unsigned (30 downto 0);

      variable floorTexX : unsigned (5 downto 0);
      variable floorTexY : unsigned (5 downto 0);

      variable temp_mult_0 : unsigned (17 downto 0);
      variable temp_mult_1 : unsigned (17 downto 0);
```

```
      variable checker_board_pattern : unsigned (0 downto 0);

      begin
            --if (rising_edge(clk) )      then

                  -- gated inputs from VGA
                  -- gated outputs to asynchronous VGA and Texture ROM
processes
--                if (clk25 = '1') then
--                      y_local <= y;
--                      cur_row_local <= cur_row;
--
--                      tex_addr_out <= tex_addr;
--                      sideOut <= sideSig;
--                      boolOut <= boolSig;
--                      texNumOut  <= texNum;
--                      texNum2Out <=  texNum2;
--
--
--                end if;
--
                  -----------------------------------------------------
-----------
                  -- stage 1 pipeline
                  temp_mult_0 :=  unsigned(signed(cur_row sll 1)+
line_minus_h) (17 downto 0);
                  inA_0 <= std_logic_vector(temp_mult_0);
                  inB_0 <= std_logic_vector(invlineHeight);
                  texY_0 := unsigned(result_0) srl 16 ;

                  temp_mult_1 :=  unsigned(signed(cur_row sll 1)+
line_minus_h2) (17 downto 0);
                  inA_1 <= std_logic_vector(temp_mult_1);
                  inB_1 <= std_logic_vector(invlineHeight2);
                  texY_1 := unsigned(result_1) srl 16 ;

                  currentDist := DIVTABLE(to_integer(480 - y));
                  weight := currentDist * invDistWall;
                  tmp := "1000000000000" - weight(23 downto 12);

                  -----------------------------------------------------
----------


                  --stage 2 pipeline
                  currentFloorX := (weight(23 downto 12) * floorX + tmp
* tmpPosX);
                  currentFloorY := (weight(23 downto 12) * floorY + tmp
* tmpPosY);

                  floorTexX := currentFloorX(23 downto 18);
                  floorTexY := currentFloorY(23 downto 18);

                  checker_board_pattern := unsigned(currentFloorX(29
downto 24) + currentFloorY(29 downto 24))(0 downto 0);

                  if (y <= row_end +1 ) then
```

```
                        if (bool ='1' or (y >= row_mid)) then
                              boolSig <= '1';
                              tex_addr <= texNum(1 downto 0) & texY_0(5
downto 0) & texX;
                              sideSig <= side1;
                        else
                              boolSig <= '0';
                              tex_addr <= texNum2(1 downto 0) &
texY_1(5 downto 0) & texX2;

                              sideSig <= side2;
                        end if;

                        texNumSig <= texNum;
                        texNum2Sig <= texNum2;
                  else

                        if (checker_board_pattern = "1") then
                              tex_addr <= ( "11"& floorTexY &
floorTexX);
                        else
                              tex_addr <= ( "00"& floorTexY &
floorTexX);
                        end if;
                        texNumSig <= "0000";
                        texNum2Sig <= "0000";
                        sideSig <= '1';
                        boolSig <= '0';
                  end if;

--                texNumprev_2 <= texNumprev;
--                texNum2prev_2 <= texNum2prev;

                  -----------------------------------------------------
---
                  --Gated Outputs

      end process;


      process (clk)
      begin
            if (rising_edge(clk)) then
                  tex_addr_out <= tex_addr;
                  sideOut <= sideSig;
                  boolOut <= boolSig;
                  texNumOut  <= texNumSig;
                  texNum2Out <=  texNum2Sig;
            end if;
      end process;
--

--  RowStartEndGen : process (clk)
--  begin
--      if rising_edge(clk) then
--                if (line_height > SCREENHEIGHT) then
--                      Row_Start <= (others => '0');
```

```
--                      Row_End <= SCREENHEIGHTMINUS1;
--                else
--                      Row_Start <= SCREENHEIGHT_HALF - ( "0" &
line_height (8 downto 1));
--                      Row_End <= SCREENHEIGHT_HALF + ( "0" &
line_height(8 downto 1));
--                end if;
--          end if;
--  end process RowStartEndGen;

end rtl;
```

==

texture_rom.vhd

==

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity texture_rom is
port(

      --clk : in std_logic;
      tex_addr : in unsigned (13 downto 0);
--    side_in  : in std_logic;
--    texNum_in : in unsigned (3 downto 0);
--    texNum2_in : in unsigned (3 downto 0);
--    bool_in   : in std_logic;
--
--    side_out  : out std_logic;
--    texNum_out : out unsigned (3 downto 0);
--    texNum2_out : out unsigned (3 downto 0);
--    bool_out   : out std_logic;
      tex_data : out unsigned (23 downto 0)


);
end texture_rom;


architecture rtl of texture_rom is

type rom_type is array(0 to 16383) of unsigned(23 downto 0);

constant ROM: rom_type := (

--Omit some data here

x"383838",x"000070",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x
"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"
00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"0
```

```
00070",x"383838",x"383838",x"000088",x"000088",x"000088",x"000088",x"00
0088",x"000088",x"000088",x"000088",x"000088",x"000088",x"000088",x"000
088",x"000088",x"000088",x"000088",x"000088",x"000088",x"000088",x"0000
88",x"000088",x"000088",x"000088",x"000088",x"000088",x"000088",x"00008
8",x"000088",x"000088",x"383838",x"383838",x"000070",x"00007c",x"00007c
",x"00007c",x"00007c",x"00007c",x"00007c",x"00007c",x"000070",x"2c2c2c"

);

signal tex_data_sig : unsigned (7 downto 0) ;

begin

process(tex_addr)
begin

      tex_data <=ROM(to_integer(tex_addr));

end process;

end rtl;

========================================================================
```

top.vhd

```
========================================================================
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top is

  port (
    -- Clocks

    CLOCK_27,                                       -- 27 MHz
    CLOCK_50,                                       -- 50 MHz
    EXT_CLOCK : in std_logic;                       -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);          -- Push buttons
    SW : in std_logic_vector(17 downto 0);          -- DPDT switches

    -- LED displays
```

```
    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment
displays
       : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs

    -- RS-232 interface

    UART_TXD : out std_logic;                      -- UART transmitter
    UART_RXD : in std_logic;                       -- UART receiver

    -- IRDA interface

--    IRDA_TXD : out std_logic;                      -- IRDA
Transmitter
    IRDA_RXD : in std_logic;                       -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                     -- Low-byte Data
Mask
    DRAM_UDQM,                                     -- High-byte Data
Mask
    DRAM_WE_N,                                     -- Write Enable
    DRAM_CAS_N,                                    -- Column Address
Strobe
    DRAM_RAS_N,                                    -- Row Address
Strobe
    DRAM_CS_N,                                     -- Chip Select
    DRAM_BA_0,                                     -- Bank Address 0
    DRAM_BA_1,                                     -- Bank Address 0
    DRAM_CLK,                                      -- Clock
    DRAM_CKE : out std_logic;                      -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);      -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
    FL_WE_N,                                         -- Write Enable
    FL_RST_N,                                        -- Reset
    FL_OE_N,                                         -- Output Enable
    FL_CE_N : out std_logic;                         -- Chip Enable

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18
Bits
    SRAM_UB_N,                                     -- High-byte Data
Mask
    SRAM_LB_N,                                     -- Low-byte Data
Mask
    SRAM_WE_N,                                     -- Write Enable
    SRAM_CE_N,                                     -- Chip Enable
    SRAM_OE_N : out std_logic;                     -- Output Enable
```

```
    -- USB controller

    OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
    OTG_ADDR : out std_logic_vector(1 downto 0);    -- Address
    OTG_CS_N,                                       -- Chip Select
    OTG_RD_N,                                       -- Write
    OTG_WR_N,                                       -- Read
    OTG_RST_N,                                      -- Reset
    OTG_FSPEED,                     -- USB Full Speed, 0 = Enable, Z =
Disable
    OTG_LSPEED : out std_logic;     -- USB Low Speed, 0 = Enable, Z =
Disable
    OTG_INT0,                                       -- Interrupt 0
    OTG_INT1,                                       -- Interrupt 1
    OTG_DREQ0,                                      -- DMA Request 0
    OTG_DREQ1 : in std_logic;                       -- DMA Request 1
    OTG_DACK0_N,                                    -- DMA Acknowledge
0
    OTG_DACK1_N : out std_logic;                    -- DMA Acknowledge
1

    -- 16 X 2 LCD Module

    LCD_ON,                     -- Power ON/OFF
    LCD_BLON,                   -- Back Light ON/OFF
    LCD_RW,                     -- Read/Write Select, 0 = Write, 1 =
Read
    LCD_EN,                     -- Enable
    LCD_RS : out std_logic;     -- Command/Data Select, 0 = Command, 1
= Data
    LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

    -- SD card interface

    SD_DAT,                     -- SD Card Data
    SD_DAT3,                    -- SD Card Data 3
    SD_CMD : inout std_logic;   -- SD Card Command Signal
    SD_CLK : out std_logic;     -- SD Card Clock

    -- USB JTAG link

    TDI,                        -- CPLD -> FPGA (data in)
    TCK,                        -- CPLD -> FPGA (clk)
    TCS : in std_logic;         -- CPLD -> FPGA (CS)
    TDO : out std_logic;        -- FPGA -> CPLD (data out)

    -- I2C bus

    I2C_SDAT : inout std_logic; -- I2C Data
    I2C_SCLK : out std_logic;   -- I2C Clock

    -- PS/2 port

    PS2_DAT,                    -- Data
    PS2_CLK : in std_logic;     -- Clock
```

```
    -- VGA output

    VGA_CLK,                                                -- Clock
    VGA_HS,                                                 -- H_SYNC
    VGA_VS,                                                 -- V_SYNC
    VGA_BLANK,                                              -- BLANK
    VGA_SYNC : out std_logic;                               -- SYNC
    VGA_R,                                                  -- Red[9:0]
    VGA_G,                                                  -- Green[9:0]
    VGA_B : out unsigned(9 downto 0);                       -- Blue[9:0]

    --  Ethernet Interface

    ENET_DATA : inout std_logic_vector(15 downto 0);    -- DATA bus
16Bits
    ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                              -- Chip Select
    ENET_WR_N,                                              -- Write
    ENET_RD_N,                                              -- Read
    ENET_RST_N,                                             -- Reset
    ENET_CLK : out std_logic;                               -- Clock 25 MHz
    ENET_INT : in std_logic;                                -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                          -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                              -- ADC Data
    AUD_DACLRCK : inout std_logic;                          -- DAC LR Clock
    AUD_DACDAT : out std_logic;                             -- DAC Data
    AUD_BCLK : inout std_logic;                             -- Bit-Stream
Clock
    AUD_XCK : out std_logic;                                -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                                      -- H_SYNC
    TD_VS : in std_logic;                       -- V_SYNC
    TD_RESET : out std_logic;                   -- Reset

    -- General-purpose I/O

    GPIO_0,                                       -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end top;

architecture datapath of top is

      component de2_i2c_av_config is
        port (
             iCLK : in std_logic;
             iRST_N : in std_logic;
             I2C_SCLK : out std_logic;
             I2C_SDAT : inout std_logic
        );
```

```
end component;

signal clk_dram : std_logic := '0';
signal clk_sys : std_logic := '0';
signal clk25 : std_logic := '0';
signal clk25_shift : std_logic := '1';
signal reset_n : std_logic := '0';
signal VGA_BLANK_SIG : std_logic := '0';
signal counter : unsigned(15 downto 0);

signal data_out    : STD_LOGIC_VECTOR (255 downto 0);
signal mem_out        : STD_LOGIC_VECTOR (255 downto 0);
signal data_out1    : STD_LOGIC_VECTOR (63 downto 0);
signal data_out2    : STD_LOGIC_VECTOR (255 downto 0);
signal Cur_Col     : unsigned (9 downto 0);

signal Cur_Row        : unsigned (9 downto 0);
signal cur_row_from_mem : unsigned (9 downto 0);
signal tex_addr : unsigned (13 downto 0);
signal ctrl                 : std_logic := '0';

signal row_s     : unsigned (8 downto 0);
signal row_e     : unsigned (8 downto 0);
signal tex_rom_out   : unsigned (23 downto 0);
signal flr_rom_out   : unsigned (7 downto 0);
signal frame_rate : unsigned (7 downto 0);
signal state_out : std_LOGIC_VECTOR(11 downto 0);

signal sky_out   : STD_LOGIC_VECTOR (7 downto 0);

 signal sram_mux            : std_logic;

 signal data_rdy            : std_logic;
    signal isSide                 : std_logic;
    signal isSide2                : std_logic;
    signal sideOut       : std_logic;
    signal write_en         : std_logic;
    signal bool          : std_logic;

    signal boolToTex     : std_logic;
    signal texNumToTex   : unsigned (3 downto 0);
    signal texNum2ToTex  : unsigned (3 downto 0);
    signal sideToTex     : std_logic;

    signal boolOut       : std_logic;
    signal texNum                 : unsigned (3 downto 0);
    signal texNum2       : unsigned (3 downto 0);
    signal texNumOut     : unsigned (3 downto 0);
    signal texNum2Out    : unsigned (3 downto 0);
    signal texX                   : unsigned (31 downto 0);
    signal texX2                  : unsigned (31 downto 0);
    signal floorX                 : unsigned (31 downto 0);
    signal floorY                 : unsigned (31 downto 0);
    signal line_minus_h     : unsigned (31 downto 0);
    signal invline                : unsigned (31 downto 0);
    signal line_minus_h2    : unsigned (31 downto 0);
    signal invline2      : unsigned (31 downto 0);
```

```
      signal invdist_out      : unsigned (31 downto 0);
      signal drawStart        : unsigned (31 downto 0);
      signal drawMid          : unsigned (31 downto 0);
      signal drawEnd                : unsigned (31 downto 0);
      signal countout      : unsigned (31 downto 0);
      signal countout2     : unsigned (31 downto 0);
      signal colAddrOut       : unsigned (9 downto 0);
      signal floor_pixel      : unsigned (11 downto 0);
      signal tmpPosX          : unsigned (31 downto 0);
      signal tmpPosY             : unsigned (31 downto 0);

      signal ledSound      : std_LOGIC_VECTOR (15 downto 0);

      --FIFO signals
      signal rdempty_sig : std_logic;
      signal wrfull_sig : std_logic;
      signal rdreq_sig : std_logic;
      signal  VGA_BLANK_OUT  : std_logic;
      signal q_fifo  : std_logic_vector (255 downto 0);

begin

--  process (clk_sys)
--  begin
--    if rising_edge(clk_sys) then
--      clk25 <= not clk25;
--          clk25_shift <= not clk25_shift;
--    end if;
--  end process;

  process (clk_sys)
  begin
    if counter = x"ffff" then
      reset_n <= '1';
    else
      reset_n <= '0';
      counter <= counter + 1;
    end if;
  end process;



  V0: entity work.sdram_pll port map (
          inclk0 => CLOCK_50,
          c0    => clk_dram,
          c1    => clk_sys,
          c2    => clk25
      );

--    V7: entity work.pll_25 port map (
--
--          inclk0 => clk_sys,
--          c0 => clk25
--
--    );

  V1: entity work.new_doom port map (
```

```
 -- 1) global signals:
     reset_n => reset_n,
     clk_0 => clk_sys,


       -- the_niosInterface_0
        ctrl_from_the_niosInterface_1_0 => ctrl,
     hardware_data_to_the_niosInterface_1_0 => ("0000000000000000000"
& std_logic_vector(frame_rate) & "000" & data_rdy),
     nios_data_from_the_niosInterface_1_0 =>  data_out,

  -- the_sram
    SRAM_ADDR_from_the_skygen_0 => SRAM_ADDR,
    SRAM_CE_N_from_the_skygen_0 => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_skygen_0 => SRAM_DQ,
    SRAM_LB_N_from_the_skygen_0 => SRAM_LB_N,
    SRAM_OE_N_from_the_skygen_0 => SRAM_OE_N,
    SRAM_UB_N_from_the_skygen_0 => SRAM_UB_N,
    SRAM_WE_N_from_the_skygen_0 => SRAM_WE_N,

-- SDRAM
      zs_addr_from_the_sdram_0 => DRAM_ADDR(11 downto 0),
      zs_ba_from_the_sdram_0(0) => DRAM_BA_0,
      zs_ba_from_the_sdram_0(1) => DRAM_BA_1,
      zs_cas_n_from_the_sdram_0 => DRAM_CAS_N,
      zs_cke_from_the_sdram_0 => DRAM_CKE,
      zs_cs_n_from_the_sdram_0 => DRAM_CS_N,
      zs_dq_to_and_from_the_sdram_0 => DRAM_DQ(15 downto 0),
      zs_dqm_from_the_sdram_0(0) => DRAM_LDQM,
      zs_dqm_from_the_sdram_0(1) => DRAM_UDQM,
      zs_ras_n_from_the_sdram_0 => DRAM_RAS_N,
      zs_we_n_from_the_sdram_0 => DRAM_WE_N,
-- New
      Cur_Row_in_to_the_skygen_0 =>  STD_LOGIC_VECTOR(Cur_Row(9 downto
0)),
      FB_angle_in_to_the_skygen_0 => STD_LOGIC_VECTOR(mem_out(169
downto 160)),   --Need to Modify !!!!
      Sky_pixel_from_the_skygen_0 => sky_out(7 downto 0),
      Sram_mux_out_from_the_skygen_0 => sram_mux,

   -- PS2
    PS2_Clk_to_the_de2_ps2_1 => PS2_CLK,
    PS2_Data_to_the_de2_ps2_1 => PS2_DAT,

        -- the_sound_controller_0
        AUD_ADCLRCK_from_the_sound_controller_0 => AUD_ADCLRCK,
     AUD_BCLK_to_and_from_the_sound_controller_0 => AUD_BCLK,
     AUD_DACDAT_from_the_sound_controller_0 => AUD_DACDAT,
     AUD_DACLRCK_from_the_sound_controller_0 => AUD_DACLRCK,
     AUX_XCK_from_the_sound_controller_0 => AUD_XCK,
        AUD_ADCDAT_to_the_sound_controller_0 => AUD_ADCDAT,
        led_from_the_sound_controller_0  => ledSound,--LEDR

         -- the_tri_state_bridge_0_avalon_slave
        address_to_the_cfi_flash_0              => FL_ADDR,
        data_to_and_from_the_cfi_flash_0 => FL_DQ,
        read_n_to_the_cfi_flash_0               => FL_OE_N,
```

```
       select_n_to_the_cfi_flash_0       => FL_CE_N,
       write_n_to_the_cfi_flash_0              => FL_WE_N

 );


 V2: entity work.de2_vga_raster port map (
   reset          => '0',
   clk            => clk25,
      bool    => boolOut,
   VGA_CLK        => VGA_CLK,
   VGA_HS         => VGA_HS,
   VGA_VS         => VGA_VS,
   VGA_BLANK      => VGA_BLANK,
      VGA_BLANK_SIG => VGA_BLANK_SIG,
   VGA_SYNC       => VGA_SYNC,
   VGA_R          => VGA_R,
   VGA_G          => VGA_G,
   VGA_B          => VGA_B,
      is_Side    => sideOut,
      Row_Start  => unsigned(mem_out(25 downto 17)),
      Row_Mid    => unsigned(mem_out(186 downto 178)),
      Row_End    => unsigned(mem_out(16 downto 8)),
      Col_Color_sky => unsigned(sky_out),    --Need to Modify
      Col_Color  => tex_rom_out,
      texNum     => texNumOut,
      texNum2    => texNum2Out,
--    Flr_Color  => flr_rom_out,
      Cur_Row          => Cur_Row,
      Cur_Col    => Cur_Col
 );

  V3: entity work.tex_gen port map (
   reset => '0',
   clk   => clk25,        -- Should be 50 MHz
      --clk25 => clk25,
      bool => mem_out(223),
      boolOut => boolOut,
      Cur_Row          => "00" & Cur_Row_from_mem,
      side1 => mem_out(63),
      side2 => mem_out(62),
      texNumOut => texNumOut,
      texNum2Out => texNum2Out,
      texNum => unsigned(mem_out(227 downto 224)),
      texNum2 => unsigned(mem_out(231 downto 228)),
      line_minus_h  => signed(mem_out(61 downto 44)),
      line_minus_h2  => signed(mem_out(222 downto 205)),
      invLineHeight => unsigned(mem_out(43 downto 26)),
      invLineHeight2 => unsigned(mem_out(204 downto 187)),
      texX                  => unsigned(mem_out(7 downto 2)),
      texX2                 => unsigned(mem_out(177 downto 172)),
      tex_addr_out     => tex_addr,
      sideOut    => sideOut,
      Row_End    => unsigned(mem_out(16 downto 8)),
      Row_Mid    => unsigned(mem_out(186 downto 178)),
      floorX                      => unsigned(mem_out(81 downto 64)),
     floorY                       => unsigned(mem_out(99 downto 82)),
```

```
      tmpPosX                       => unsigned(mem_out(117 downto
100)),
      tmpPosY                       => unsigned(mem_out(135 downto
118)),
      invDistWall             => unsigned(mem_out(147 downto 136)),
      y                                   => Cur_Row_from_mem(8 downto
0)

  );

   V4: entity work.texture_rom port map (
    --clk         => clk_sys,
       tex_addr => tex_addr,
       tex_data => tex_rom_out

  );

  V9: ENTITY work.FIFO port map(
            data        => x"FFF" &
                                          "0" &
                                          VGA_BLANK_OUT &
                                          std_logic_vector (colAddrOut)
&
                                          std_logic_vector(texNum2) &
                                          std_logic_vector(texNum) &
                                          bool &

      std_logic_vector(line_minus_h2(17 downto 0)) &
                                          std_logic_vector(invline2(17
downto 0)) &
                                          std_logic_vector(drawMid(8
downto 0)) &
                                          std_logic_vector(texX2(5
downto 0)) &
                                          "00" &
                                          std_LOGIC_VECTOR(data_out(169
downto 160)) &
                                          x"FFF"  &

      STD_LOGIC_VECTOR(invdist_out(11 downto 0)) &
                                          STD_LOGIC_VECTOR(tmpPosY(17
downto 0)) &
                                          STD_LOGIC_VECTOR(tmpPosX(17
downto 0)) &
                                          STD_LOGIC_VECTOR(floorY(17
downto 0)) &
                                          STD_LOGIC_VECTOR(floorX(17
downto 0)) &
                                          isSide & isSide2 &

      STD_LOGIC_VECTOR(line_minus_h(17 downto 0)) &
                                          STD_LOGIC_VECTOR(invline(17
downto 0)) &
                                          STD_LOGIC_VECTOR(drawStart(8
downto 0)) &
                                          STD_LOGIC_VECTOR(drawEnd(8
downto 0)) &
```

```
                                          STD_LOGIC_VECTOR(texX(5
downto 0)) &
                                          "00",
            rdclk       => clk25,
            rdreq       => rdreq_sig,
            wrclk       =>  clk_sys,
            wrreq       =>  write_en,
            q           => q_fifo,
            rdempty      => rdempty_sig ,
            wrfull      => wrfull_sig
      );

      rdReqGen: process (rdempty_sig)

      begin

            rdreq_sig <= not rdempty_sig;

      end process rdReqGen;


  V5: entity work.memcustom port map (
            clock             => clk25,
--          data              => data_out,
            data              => q_fifo,
            rdaddress   => Cur_Col,
            rd_req      => rdreq_sig,
--          wraddress   => colAddrOut,
--          wraddress   =>  data_out(255 downto 246),
            --wren                  => write_en,
      --VGA_BLANK   => VGA_BLANK_SIG,
            row_in      => cur_row,
            row_out     => cur_row_from_mem,
--          wren              => ctrl,
            q                       => mem_out
      );


       V6: entity work.framerate_calc port map (
             clk              => clk_sys,       -- Should be 50 MHz
             wr_addr    => unsigned(data_out(255 downto 246)),

             frame_rate => frame_rate
      );


--    V7: entity work.ray_FSM port map (
--                      clk               => clk_sys,
--                      VGA_BLANK   => VGA_BLANK_SIG,
--                      control           => ctrl,
----                    reset             => data_out(240),
--                      posX              => unsigned(data_out(31
downto 0)),
--                      posY              => unsigned(data_out(63
downto 32)),
--                      countstep   => unsigned(data_out(95 downto
64)),
```

```
--                      rayDirX           => signed(data_out(127 downto
96)),
--                      rayDirY           => signed(data_out(159 downto
128)),
--                      colAddrIn   => unsigned(data_out(255 downto
246)),
--                      isSide            => isSide,
--                      texNum            => texNum,
--                      texX              => texX,
--                      floorX            => floorX,
--                      floorY            => floorY,
--                      tmpPosXout  => tmpPosX,
--                      tmpPosYout  => tmpPosY,
--                      countout          => countout,
--                      line_minus_h => line_minus_h,
--                      invline      => invline,
--                      invdist_out  => invdist_out,
--                      drawStart    => drawStart,
--                      drawEnd      => drawEnd,
--                      colAddrOut   => colAddrOut,
--                      state_out    => state_out,
--                      WE          => write_en,
--                      ready             =>   data_rdy
--    );
--
      V8: entity work.ray_FSM port map (
                        clk           => clk_sys,
                        control       =>  ctrl,
                        VGA_BLANK     => VGA_BLANK_SIG,
                        VGA_BLANK_OUT => VGA_BLANK_OUT,
                        wrfull      => wrfull_sig ,
                        posX                => unsigned(data_out(31
downto 0)),
                        posY                => unsigned(data_out(63
downto 32)),
                        countstep     => unsigned(data_out(95 downto
64)),
                        rayDirX             => signed(data_out(127
downto 96)),
                        rayDirY             => signed(data_out(159
downto 128)),
                        colAddrIn     => unsigned(data_out(255 downto
246)),
                        tmpPosXout    => tmpPosX,
                        tmpPosYout    => tmpPosY,
                        isSide        => isSide,
                        isSide2       => isSide2,
                        bool          => bool,
                        texNum        => texNum,
                        texNum2       => texNum2,
                        texX          => texX,
                        texX2         => texX2,
                        floorX        => floorX,
                        floorY        => floorY,
                        countout            => countout,
                        countout2     => countout2,
                        line_minus_h  => line_minus_h,
```

```
                                invline       => invline,
                                line_minus_h2 => line_minus_h2,
                                invline2      => invline2,
                                invdist_out   => invdist_out,
                                drawStart     => drawStart,
                                drawMid       => drawMid,
                                drawEnd       => drawEnd,
                                colAddrOut    => colAddrOut,
                                WE            => write_en,
                                ready               => data_rdy,
                                state_out     => state_out

 );

  i2c : de2_i2c_av_config port map (
    iCLK     => clk_sys,
    iRST_n   => '1',
    I2C_SCLK => I2C_SCLK,
    I2C_SDAT => I2C_SDAT
  );


  HEX7     <= "0001001"; -- Leftmost
  HEX6     <= "0000110";
  HEX5     <= "1000111";
  HEX4     <= "1000111";
  HEX3     <= "1000000";
  HEX2     <= (others => '1');
  HEX1     <= (others => '1');
  HEX0     <= (others => '1');          -- Rightmost
  LEDG     <= (others => '1');
  LEDR     <=  state_out & "000000";
  LCD_ON   <= '1';
  LCD_BLON <= '1';
  LCD_RW <= '1';
  LCD_EN <= '0';
  LCD_RS <= '0';

  SD_DAT3 <= '1';
  SD_CMD <= '1';
  SD_CLK <= '1';

  UART_TXD <= '0';
  --DRAM_ADDR <= (others => '0');
  --DRAM_LDQM <= '0';
  --DRAM_UDQM <= '0';
  --DRAM_WE_N <= '1';
  --DRAM_CAS_N <= '1';
  --DRAM_RAS_N <= '1';
  --DRAM_CS_N <= '1';
  --DRAM_BA_0 <= '0';
  --DRAM_BA_1 <= '0';
  DRAM_CLK <= clk_dram;
  --DRAM_CKE <= '0';
  --FL_ADDR <= (others => '0');
  --FL_WE_N <= '1';
  FL_RST_N <= '1';
```

```
  --L_OE_N <= '1';
  --FL_CE_N <= '1';
  OTG_ADDR <= (others => '0');
  OTG_CS_N <= '1';
  OTG_RD_N <= '1';
  OTG_RD_N <= '1';
  OTG_WR_N <= '1';
  OTG_RST_N <= '1';
  OTG_FSPEED <= '1';
  OTG_LSPEED <= '1';
  OTG_DACK0_N <= '1';
  OTG_DACK1_N <= '1';

  TDO <= '0';

  ENET_CMD <= '0';
  ENET_CS_N <= '1';
  ENET_WR_N <= '1';
  ENET_RD_N <= '1';
  ENET_RST_N <= '1';
  ENET_CLK <= '0';

  TD_RESET <= '0';

  --I2C_SCLK <= '1';

  --AUD_DACDAT <= '1';
  --AUD_XCK <= '1';

  -- Set all bidirectional ports to tri-state
  DRAM_DQ     <= (others => 'Z');
  --FL_DQ       <= (others => 'Z');
  SRAM_DQ     <= (others => 'Z');
  OTG_DATA    <= (others => 'Z');
  LCD_DATA    <= (others => 'Z');
  SD_DAT      <= 'Z';
  --I2C_SDAT    <= 'Z';
  ENET_DATA   <= (others => 'Z');
  --AUD_ADCLRCK <= 'Z';
  --AUD_DACLRCK <= 'Z';
  --AUD_BCLK    <= 'Z';
  GPIO_0      <= (others => 'Z');
  GPIO_1      <= (others => 'Z');

end datapath;
========================================================================
```

helloworld.c

```
========================================================================
#include <math.h>
#include <io.h>
#include <system.h>
#include <stdio.h>
#include "sky.h"
```

```
//#define lookupLength 3480
//#define RAD 0.0018055130193045
//#define HALF_RAD 0.00090275650965225

#define lookupLength 4096
#define RAD 0.0015339807878856
#define HALF_RAD 0.0007669903939428

//#define lookupLength 3072
//#define RAD 0.0020453077171808
//#define HALF_RAD 0.0010226538585904

#define screenWidth 640
#define halfScreenWidth 320
#define screenHeight 480
#define texWidth 64
#define texHeight 64
#define mapWidth 32
#define mapHeight 32
#define extensionFactor 5
#define loopbackFactor 4

#define RED_8BIT 0xE0 //red
#define GREEN_8BIT 0x1C //green
#define BLUE_8BIT 0x03 //blue
#define YELLOW_8BIT 0xFC //yellow
#define WHITE_8BIT 0xFF //white
#define VIOLET_8BIT 0xE2 //violet

#define CTRL_WRITE_HIGH 0
#define CTRL_WRITE_LOW 0

#define posShift 22

#define IOWR_RAM_DATA(base, offset, data) \
IOWR_32DIRECT(base, (offset*4), data)

#define IORD_RAM_DATA(base, offset) \
IORD_32DIRECT(base, (offset*4) )

#define IOSKYWR_RAM_DATA(base, offset, data) \
  IOWR_16DIRECT(base, (offset) * 2, data)

char worldMap[mapWidth][mapHeight]=
{
 9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
 9,0,0,7,0,8,0,8,0,0,8,7,0,2,5,2,6,2,5,2,0,6,0,0,0,0,0,0,0,0,0,9,
 9,2,3,3,0,0,0,0,0,8,8,4,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,6,0,9,
 9,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,9,
 9,2,3,3,0,0,0,0,0,8,8,4,0,0,0,0,0,0,0,0,0,0,0,6,0,0,0,6,0,0,9,
 9,0,0,0,0,0,0,0,0,0,8,4,0,0,0,0,0,6,6,6,0,6,7,0,0,0,5,0,0,0,0,9,
 9,8,8,8,0,8,8,8,8,8,8,4,4,4,4,4,4,6,0,0,0,0,0,0,0,0,0,0,0,7,0,9,
 9,6,7,7,0,7,7,7,7,0,8,0,8,0,8,0,8,4,0,4,0,6,7,0,0,0,0,0,0,0,0,9,
 9,7,0,0,0,0,0,0,7,8,0,8,0,8,0,8,8,6,0,0,0,0,0,0,0,8,0,8,0,0,9,
 9,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,9,
 9,2,0,0,0,0,0,0,0,0,0,0,0,0,0,8,6,0,6,0,6,0,0,0,0,0,0,0,0,0,9,
 9,7,0,0,0,0,0,0,7,8,0,8,0,8,0,8,8,6,4,6,0,6,6,0,0,0,0,0,0,0,0,9,
 9,2,7,7,0,7,7,7,7,8,8,4,0,6,8,4,8,3,3,3,0,3,6,0,0,0,0,0,0,0,0,9,
 9,6,2,2,0,2,2,2,2,4,6,4,0,0,6,0,6,3,0,0,0,0,0,0,0,0,0,0,0,0,0,9,
```

```
 9,6,0,0,0,0,0,2,2,4,0,0,0,0,0,0,4,3,0,0,0,0,6,0,0,0,0,0,0,0,0,9,
 9,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,4,3,0,0,0,0,0,0,3,2,0,4,3,0,0,9,
 9,7,0,0,0,0,0,0,1,4,4,0,4,4,6,0,6,3,3,0,0,0,0,0,0,0,0,0,0,0,0,9,
 9,0,0,0,0,0,0,0,2,2,2,0,2,2,2,6,6,0,0,5,0,5,0,0,0,0,0,0,0,0,0,9,
 9,7,0,0,0,0,0,2,2,2,0,0,0,2,2,3,5,0,5,0,0,0,5,0,0,0,0,0,0,0,0,9,
 9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,9,
 9,7,0,0,0,0,0,0,3,4,2,1,1,2,0,4,4,3,1,2,2,0,0,0,0,0,0,0,0,0,0,9,
 9,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,7,2,7,2,2,2,0,7,0,8,8,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,7,2,7,2,2,2,0,7,0,8,8,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,0,0,0,2,2,0,7,0,8,8,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,2,0,0,0,0,0,7,0,8,8,0,3,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,2,0,2,2,2,0,2,0,8,8,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,2,0,0,0,1,0,2,0,3,3,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,2,7,2,0,2,0,1,0,8,8,0,3,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,2,0,2,7,1,0,1,0,1,0,8,8,0,5,0,4,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,2,0,0,0,2,7,2,1,2,0,7,0,0,0,0,5,0,0,0,7,0,0,0,0,0,0,0,0,0,0,9,
 9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
};

/// INTERRUPT VARIABLES
volatile int offset=0;
volatile alt_u8 new = 128;

static void  note_isr (void * context, alt_u32 id){
	//if(offset>=14531){
	if(offset>=227327){
		offset=0;
	}
	else{
		offset++;
	}

	new = IORD_8DIRECT(CFI_FLASH_0_BASE,offset);
	IOWR_8DIRECT(SOUND_CONTROLLER_0_BASE,0,new);

	return;

}

//fixed point conversion functions
int doubleToInt(double);


void DrawTexture(unsigned int columnIndex, unsigned int texX, unsigned int
rowStart, unsigned int rowEnd, unsigned int side, unsigned int texNum,
unsigned int invLineHeight , unsigned int line_minus_h);
void DrawAccelerate(int angle, int posX, int posY, int countstep, int
rayDirX, int rayDirY, unsigned int columnIndex);
int absVal(int v);

///UNECESSARY CAN BE IMPLEMENTED WITHOUT THESE TWO LOOKUP
int dirsine[lookupLength];
int dircosine[lookupLength];

/// MEMORY HERE CAN BE CUT DOWN IF NECESSARY
int sine[lookupLength];
int cosine[lookupLength];
```

```
int main()
{

	IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 0, 0);
	int dir = 0;
	int posX = doubleToInt(21.5), posY = doubleToInt(11.5);  //x and y
start position
	 int x =0;
	 int p, q;


  IOSKYWR_RAM_DATA(SKYGEN_0_BASE, 262143, 0x0000);

  for(p = 0; p < 480; p++){
	 for(q = 0; q < 512; q++)
	 {

		IOSKYWR_RAM_DATA(SKYGEN_0_BASE, p*512+q,
(sky[p*1024+q*2+1]<<8) + (sky[p*1024+q*2]));

	 }
  }

  IOSKYWR_RAM_DATA(SKYGEN_0_BASE, 262143, 0x000F);
  double sine_temp;
  double cosine_temp;

	for(x = 0; x < lookupLength ; x++)
	{
		//calculate ray position and direction
		sine_temp = sin(x*RAD + HALF_RAD);
		cosine_temp = cos(x*RAD + HALF_RAD);

		dirsine[x] = doubleToInt(sin(x*RAD));
		dircosine[x] = doubleToInt(cos(x*RAD));
		sine[x] = doubleToInt(sine_temp)>>extensionFactor;
		cosine[x] = doubleToInt(cosine_temp)>>extensionFactor;
	}

  int angle;
  int fish_angle;
  int move;

  int rayDirX;
  int rayDirY;
  int count_step;

  int k;
  int forward = 0;
  int backward = 0;
  int left = 0;
  int right = 0;

  unsigned char code = 0;
  int hardwareData = 0;


  alt_irq_register(SOUND_CONTROLLER_0_IRQ ,NULL,(void*) note_isr);

  while(1)
```

```
{

    x = 0;
  for(k = -halfScreenWidth; k < halfScreenWidth; k++)
  {

      if ( (k & 0xF) == 0)
      {
            code = 0;
            code = IORD_8DIRECT(DE2_PS2_1_BASE, 1);

            switch(code)
            {
                  case 'u':
                        forward = 1;
                        backward = 0;
                        break;
                  case 'r':
                        forward = 0;
                        backward = 1;
                        break;
                  case 't':
                        right = 1;
                        left = 0;
                        break;
                  case 'k':
                        left = 1;
                        right = 0;
                        break;
                  case 'U':
                        forward = 0;
                        break;
                  case 'R':
                        backward = 0;
                        break;
                  case 'T':
                        right = 0;
                        break;
                  case 'K':
                        left = 0;
                        break;
                  case ')':
                      forward = 0;
                      backward = 0;
                      right = 0;
                      left = 0;
                      break;
            }
     }

      angle = (dir + k)&0xFFF;

      fish_angle = k&0xFFF;

    //calculate ray position and direction

    rayDirX = cosine[angle];
    rayDirY = sine[angle];
    count_step = cosine[fish_angle];
```

```
     DrawAccelerate(angle, posX, posY, count_step, rayDirX, rayDirY, x);
     IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 0, 0);

     hardwareData = IORD_32DIRECT(NIOSINTERFACE_1_0_BASE, 1);

        while (!(hardwareData & 1)){
               hardwareData = IORD_32DIRECT(NIOSINTERFACE_1_0_BASE, 1);
         }

      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 0, 0xFFFFFFFF);

      x++;
    }

    //move forward if no wall in front of you
    if (forward == 1)
    {
        move = dircosine[dir]>>4;
        if(worldMap[(posX + move)>>posShift][posY>>posShift] == 0)
                posX += move;

        move = dirsine[dir]>>4;
        if(worldMap[posX>>posShift][(posY+move)>>posShift] == 0)
                posY += move;
    }
    //move backwards if no wall behind you
    if (backward == 1)
    {
        move = dircosine[dir]>>4;
        if(worldMap[(posX - move)>>posShift][posY>>posShift] == 0)
                posX -= move;

        move = dirsine[dir]>>4;
        if(worldMap[posX>>posShift][(posY - move)>>posShift] == 0)
                posY -= move;
    }
    //rotate to the right
    if (right == 1)
    {
      //both camera direction and camera plane must be rotated
       dir = (dir +13)&0xFFF;
    }
    //rotate to the left
    if (left == 1)
    {
      //both camera direction and camera plane must be rotated
        dir = (dir - 13)&0xFFF;
    }
  }
    return 0;
}

int absVal(int v)
{
  return v * ((v>0) - (v<0));
}

int intToDouble(int a)
{
    return a>>posShift;
```

```
}

int doubleToInt(double a)
{
    return (int)(a*(1<<posShift));
}

void DrawTexture(unsigned int columnIndex, unsigned int texX, unsigned int
rowStart, unsigned int rowEnd, unsigned int side, unsigned int texNum,
unsigned int invLineHeight , unsigned int line_minus_h)
{
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 8, (columnIndex  << 22));
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 1, ((texNum & 3) + (texX <<2) +
(rowEnd<< 8) + (rowStart<<17)+ ((invLineHeight & 0x3F)<<26)));
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 2, ((invLineHeight & 0x3FFFF)
>> 6) + ((line_minus_h & 0x3FFFF) << 12 ) + (side << 31));
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 0, 1);
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 0, 0);
}

void DrawAccelerate(int angle, int posX, int posY, int countstep, int
rayDirX, int rayDirY, unsigned int columnIndex)
{
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 8, (columnIndex << 22));
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 1, posX);
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 2, posY );
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 3, countstep );
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 4, rayDirX);
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 5, rayDirY );
      IOWR_RAM_DATA(NIOSINTERFACE_1_0_BASE, 6, angle & 0x03FF);
}
===========================================================
```

readwav.m

```
============================================================
clear all;
fp=fopen('F:\Embedded System\music\msg.txt','w');
fid=fopen('F:\Embedded System\music\sound.bin','wb');
[s,sr,n]=wavread('F:\Embedded System\music\intwap.wav');
[s1,sr1,n1]=wavread('F:\Embedded System\music\msg.wav');

data=s*(2^8/2)+(2^8/2);
data=floor(data);

data1=s1*(2^8/2)+(2^8/2);
data1=floor(data1);

offset=1;
whole=zeros(400000,1);

for i=1:2:length(data)
    whole(offset)=data(i);
    offset=offset+1;
end
for i=1:length(data1)
```

```
    whole(offset)=data1(i);
    offset=offset+1;
end

fwrite(fid,whole,'uint8')
fclose(fid)

%sound(s)
=========================================================================
```